Escuela Politécnica Superior

Interfaz y control de UAVs

Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor:
Enrique Mas Candela

Tutor/es:
Fidel Aznar Gregori

Universitat d'Alacant
Universidad de Alicante

Junio 2018

Contents

Resum

Aquest projecte està orientat a portar a la pràctica algunes de les últimes tècniques de Deep Learning en visió artificial. En el projecte s'ha desenvolupat un sustema de control autònom per un UAV basat en el seguiment d'objectes. Durant el desenvolupament del projecte s'han analitzat i provat els sistemes de seguiment visual d'objectes de l'estat de l'art en l'objectiu de construir una aplicació mòbil en una simple interfaç d'usuari capaç de comunicar-se en l'UAV per enviar-li comandements de moviment per fer-lo seguir un objectiu autònomament. El propòsit final del projecte es ser provat en un UAV real per demostrar el seu funcionament. Actualment està sent provat.

Abstract

This project is oriented to put into practice some of the last deep learning techniques in computer vision. In the project it has been developed an autonomous control system for a UAV based on object tracking. Along the development of the project the state of art of visual object trackers will be analyzed and tested in order to finally build a mobile application with a simple user interface capable of communicating with the UAV for sending it movement commands so that making it to follow a selected target autonomously. The final purpose of this project is to be tested on a real UAV to show how it works. It is currently being tested.

1 Introduction

1.1 Motivation

Deep learning has been growing up very quickly from a few years now. Concretely in computer vision thanks to the apparition of Convolutional Neural Networks has been a huge development allowing to build very powerful models and bringing great advances to machine learning. The main motivation of this project was to take to practice some of these advances to a more practical scenario so I can test and show the potential applications of these techniques.

On the other hand, the rise of the use of UAVs in last years and the possibilities they can offer made me think about I can use any of the state of art deep learning techniques on it. I considered building an object tracking system as it could be an interesting idea to carry out. I had also to keep in mind that object tracking requires working at a high frame rates reducing latency as much as possible and would be a challenge make it work.

A great part of the motivation also lies on the fact that there is not any similar open source software right now. The most alike tools are commercial ones which are not able to be used as you want as they are subjected to the owner's specification. So this project gave me the chance to create something new.

This project also allowed me to work with new cutting edge technologies, being able to manipulate visual object trackers that were developed during the last year as well as frameworks that are even on their initial releases.

Relation of the project with Computer Science

This project joins different fields of Computer Science which can be easily related with degree subjects:

- *Intelligent Systems* or *Machine Learning*. This is the most important part of the project as it involves all the object tracking system component as well as the movement prediction. This can be related to the contents of the subjects *Sistemas Inteligentes (SI)*, *Desafíos de Programación (DP)* and *Visión Artificial y Robótica (VAR)* in the degree.

- *Robotics.* An UAV is actually a flying robot. All the functionalities of the UAV as well as how to move it and communicating with it is highly related with Robotics. This field is highly related with the degree subjects *Visión Artificial y Robótica (VAR)* and *Tecnologías y Arquitecturas Robóticas (TAR)*.

- *Software Development.* Finally all the project is wrapped in a mobile application which pertains to the field of Software Development. Software development is related to many of the degree subjects like *Diseño de Sistemas Software (DSS)* or *Herramientas Avanzadas para el Desarrollo de Aplicaciones (HADA)*.

1.2 State of art

Computer vision has always been one of the most studied fields of Artificial Intelligence. First Computer Vision was based on building complex heuristics after a feature extraction with tools such as Kalman filters [9], Harris corner detectors [6], SIFT [15], SURF [1]... This process has always been a difficult and tedious work as you have to build very complex systems for any specific task.

When Convolutional Neural Networks arrived, this process changed completely. Yann LeCun proved with LeNet [14] in 1998 the capabilities of CNN on image classification, opening a new huge range of possibilities that offered Deep Learning. However, due to the low performance capabilities of computers by that time, CNNs did not succeeded until AlexNet [13] was published in 2012. Since then, Deep Learning allowed to build real time and very robust tracker systems like GOTURN [8] in 2016, one of the first deep learning trackers able to perform at real time.

This project is oriented to use the state of art of visual tracking, using some of the most advanced deep learning techniques and the last state of art models. Along the development of the project I will expose and use some of the most advanced models including YOLO [17, 19], Re^3 [5] among other which will be explained in detail along the project and will be analyzed in a practical scenario for building with them the tracking system.

1.3 Objectives

The purpose of this project was to develop an autonomous UAV flight controller capable of tracking the position of a target selected in a user interface and move itself for following the target according its position. For carrying out this project we established the following objectives:

- *Control a UAV through software.* One of the most important tasks I would have in this project was to be able to move the UAV autonomously. I disposed of some DJI quadcopters which provide an SDK for communicating through software with the aircraft so I hat to check it out how it worked.

- *Testing the state of art visual trackers.* I had to get in the state of art of visual trackers in order to know how to build my system. I had to find a tracker that fit my needs, being able to follow arbitrary targets and that would be able to run at real time.

- *Know about the mobile deep learning frameworks.* As the SDK I was going to use work on mobile operating systems only, I would have to implement the tracker on a mobile device. For this I had to choose the most appropriate deep learning framework in order to make the model work as better as possible.

- *Port the tracker to a mobile platform.* Knowing which tracker I would use and in which platform I would implement it, I had to proceed to develop the model and test it on the mobile device.

- *Predict the movement from the target position.* Given a target position I had to elaborate a movement prediction based on that. This would define the behavior of the UAV and how it will follow the target.

- *Add the movement prediction feature to the tracker model.* Once I predicted the movement my aim was add a new regression layer to the tracker model and train it with data I would generate in order to get the movement prediction directly from the neural network, having a full tracking system embedded in a single deep learning model.

All this objectives are faced to achieve the main objective of building an autonomous UAV controller able to follow a given target knowing at all times its position and generating the proper movement for following that target.

2 UAV

An UAV (*Unmanned Aerial Vehicle*), commonly know as *drone*, is essentially a flying robot propelled by one or more motors that can is controlled by an embedded system that has on board. This system can control directly the aircraft autonomously or act as a high level interface for remote controlling. There are multiple types of UAVs classified according different criteria, from military drones to logistic or racing ones, however, in the project I will center in simple commercial drones that provide all I need.

For carrying out the project I had available a DJI Mavic Pro quadcopter. Quadcopters are the most common UAVs in market as its functioning is very simple. Quadcopters, as the name says, had four rotors placed in cross. DJI offers many tools and helping for developing including a very powerful API for communicating with the aircraft. This aircraft also disposes an HD camera sensor which I'll use for the object tracking.

Figure 1: DJI Mavic Pro.

2.1 UAV control

I had to learn a bit about some aspects of how quadcopters work, specifically hoy to control them. Generally all the aircrafts are controlled by setting up its 3D rotations. This control system locates 3 fictional axes on the drone which each of them is responsible of the rotation of the aircraft in this axis. The rotation of each axis has a different name being:

- *Pitch*: the rotation on the Y axis. Modifying the pitch you can move the quadcopter forwards and backwards.

- *Roll*: the rotation on the X axis. Modifying the roll you can move the quadcopter laterally.

- *Yaw*: the rotation on the Z axis. Modifying the pith you can rotate the quadcopter.

Also, the *throttle* controls the movement of the quadcopter on the Z axis, allowing to ascend and descend on the air.

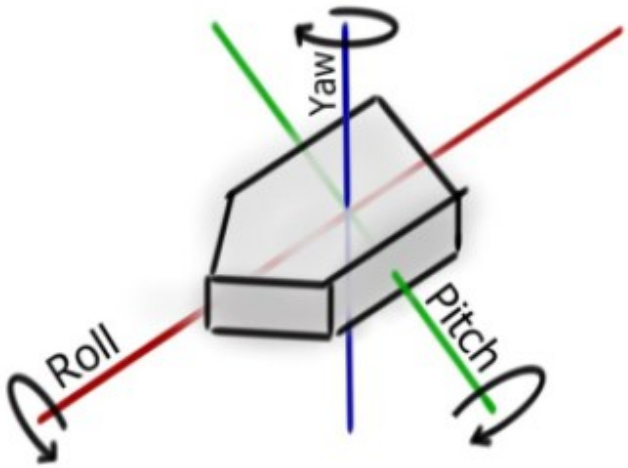


Figure 2: Aircraft axes.

2.2 DJI API

DJI provides a high level API to interact easily with the aircraft. You can fetch information from the on board computer including the information of the sensors of the device, being able to know the state of the aircraft or the environment at real time as well as sending commands to the quadcopter for performing different actions.

The DJI API is available for iOS and Android and for communicating with the aircraft you have to connect the device to the Remote Controller that acts as an interface for interacting with the quadcopter.

2.2.1 Registering an application

First of all for creating an application with the DJI SDK you have to register it. When you register the application, you are asking DJI permission for your app to be able to communicate with the UAV for preventing anyone to freely build any kind of application as it can be dangerous in some cases.

For registering an application, you first have to join the DJI Developer program. Once they have accepted you, you can request for creating a new application. For the request you will have to specify the type of SDK, bundle identifier of your application (that has to be exactly the same in you project), the platform (iOS or Android) and some information about your application: a category for it and a short description of what it does.

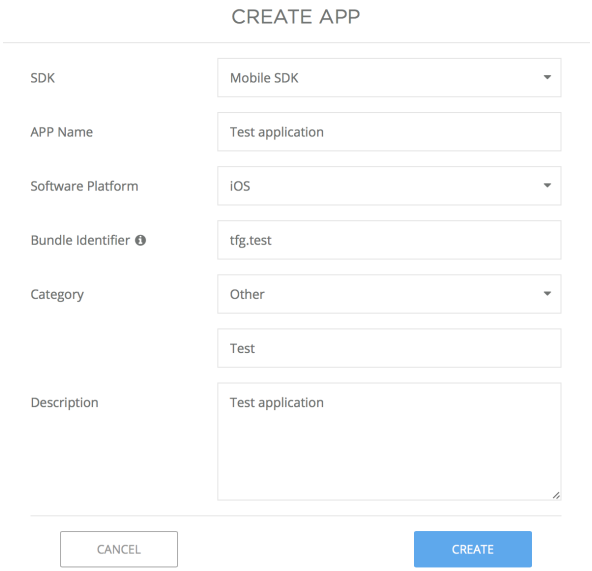
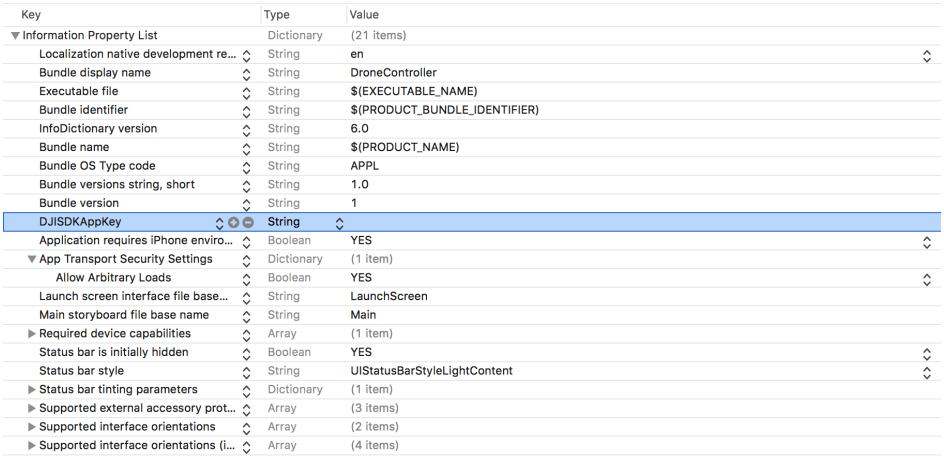


Figure 3: App registration form.

After the request, DJI will send you an email for notifying you if your app has been accepted. If it does, you will find in you DJI Developer account's page an API Key for using the SDK. With this, you can create a new application that uses the DJI SDK. You only have to create a new iOS application with the bundle identifier you specified on the application request and add a field on the file *Info.plist* with the name *DJISDKAppKey* that contains the SDK API Key that DJI provided you.



Key	Type	Value
Information Property List	Dictionary	(21 items)
Localization native development re...	String	en
Bundle display name	String	DroneController
Executable file	String	$(EXECUTABLE_NAME)
Bundle identifier	String	$(PRODUCT_BUNDLE_IDENTIFIER)
InfoDictionary version	String	6.0
Bundle name	String	$(PRODUCT_NAME)
Bundle OS Type code	String	APPL
Bundle versions string, short	String	1.0
Bundle version	String	1
DJISDKAppKey	String	
Application requires iPhone enviro...	Boolean	YES
App Transport Security Settings	Dictionary	(1 item)
Allow Arbitrary Loads	Boolean	YES
Launch screen interface file base...	String	LaunchScreen
Main storyboard file base name	String	Main
Required device capabilities	Array	(1 item)
Status bar is initially hidden	Boolean	YES
Status bar style	String	UIStatusBarStyleLightContent
Status bar tinting parameters	Dictionary	(1 item)
Supported external accessory prot...	Array	(3 items)
Supported interface orientations	Array	(2 items)
Supported interface orientations (i...	Array	(4 items)

Figure 4: Info.plist file with the API Key field.

Once the project is set up, you would have to handle the registration. The registration process is done each time the application starts and it requires internet connection. The registration will send the access credentials to DJI to check out if the application can use the SDK. For registering the application we have to make the application's controller to be a child class of the class *DJISDKManagerDelegate* and write a registration handler function:

```
func appRegisteredWithError(_ error: Error?) {
    if (error != nil) {
        let alert = UIAlertController(title: "Register app failed",
            message: "Register app failed! Please enter your " +
                "app key and check the network.",
            preferredStyle: UIAlertControllerStyle.alert)

        self.present(alert, animated: true, completion: nil)
        alert.addAction(UIAlertAction(title: "Ok",
            style: UIAlertActionStyle.default,
            handler: nil))
    } else {
        let alert = UIAlertController(title: "Register app succeeded",
            message: "Register app succeeded.",
            preferredStyle: UIAlertControllerStyle.alert)
        alert.addAction(UIAlertAction(title: "Ok",
            style: UIAlertActionStyle.default,handler: nil))
        self.present(alert, animated: true, completion: nil)
    }
    DJISDKManager.startConnectionToProduct()
}
```

And finally we have to call the following function:

```
DJISDKManager.registerApp(with: self)
```

for beginning the registration process.

2.2.2 Getting the image of the camera

One of the tasks I had to face with the SDK was the image obtention. Natively the iOS SDK does allow the developers to get the images taken by the aircraft. However, DJI provides a class for building FPV applications that do not directly gives access to the images but can be easily modified for making it capable. This class is called DJI Video-Previewer [1] and consists, as the name says, in a GUI Object that gives us a preview of what the camera is getting at real time and for allowing the access to the images of the camera in a usable format I needed to make public the private property *videoExtractor* from the *VideoPreviewer* class. At the moment I had access to the *videoExtractor* I had to make my controller to be a child class from the *DJIVideoFeedListener* and making it to be a delegate of *videoFeeder* for handling the video feeder with the current function:

```
func videoFeed(_ videoFeed: DJIVideoFeed, didUpdateVideoData rawData: Data) {
    let videoData = rawData as NSData
    let videoBuffer = UnsafeMutablePointer<UInt8>.allocate(
        capacity: videoData.length)

    videoData.getBytes(videoBuffer, length: videoData.length)
    VideoPreviewer.instance().push(
        videoBuffer, length: Int32(videoData.length))
}
```

that feeds the videoPreviewer instance with the raw data from the camera every moment. At this point, the video previewer is having full access to the camera and whenever I want I can fetch the image that is being seen by the camera in a *CVPixel-Buffer* format :

```
let image = videoPreviewer!.videoExtractor!.getCVImage().takeRetainedValue()
```

2.2.3 Sending movement commands to the UAV

After getting the images the other task I had to do was control the movement of the aircraft with the SDK. DJI provides two ways of moving the UAV according our needs:

- *DJI Missions*: a high level API where the user assign tasks to the quadcopter and it decides itself the movements (e.g. you tell the drone to go somewhere and it moves in the proper direction in function of where the target is). There are lots of kinds of missions, from going to a list of GPS locations to follow a GPS located subject among other.

[1] https://github.com/dji-sdk/Mobile-SDK-iOS/tree/master/Sample%20Code/VideoPreviewer

- *DJI Virtual Sticks Controller*: it is a low level API that send directly movement commands to the aircraft. You specify the pitch, roll, yaw and throttle of the aircraft as you were sending remote control signals. This class allows more control over the quadcopter.

As I need a low level control system for my project in order to send directly movement commands to the aircraft, I used the Virtual Sticks controller. They work in a very straightforward way, once the motors are turned on you send the virtual stick commands to the quadcopter and if all the configuration is correct (the aircraft orientation mode has to be *Aircraft Heading* and the virtual stick mode has to be enabled) the aircraft will move according the sent command. Here we have an example of sending pitch and roll commands:

```
func setVel(pitch p: Float, roll r: Float) {
    if (self.flightController!.isVirtualStickControlModeAvailable()) {
        self.log.text = String(format: "pitch: \%3.2f, roll: \%3.2f", p, r)
    } else {
        flightController!.setVirtualStickModeEnabled(true)
        flightController!.setFlightOrientationMode(
            DJIFlightOrientationMode.aircraftHeading)

        self.log.text = "Error, control mode unavailable."
    }
    let controlData = DJIVirtualStickFlightControlData(
        pitch: p,
        roll: r,
        yaw: 0,
        verticalThrottle: 0
    )
    flightController!.send(controlData)
}
```

It should be noted that the virtual stick control mode is enabled and the flight orientation mode is set to the proper one.

3 Tracking

Deep learning is becoming increasingly important in machine learning and is overcoming in many ways traditional machine learning techniques. Thanks to the quick advance of neural networks in last years they are giving as very good results in lots of different problems. Particularly in computer vision, CNNs (Convolutional Neural Networks) [13] have been very important in the development of new models with better results than most methods used before.

One of the drawbacks of deep learning techniques is the computational cost. As deep neural network models have to do even millions of computations per predictions, being difficult the use of these models on low performance devices. However, my intention was to run the tracker on the mobile phone in order to communicate it with the DJI SDK with the few delay as possible and of course, getting good results at real time. As consequence, I had to found a fast as possible model in order to fit in a mobile device. For this, I looked for in the state of art trackers the fastest models (according to benchmarks), among which I found GOTURN [8], Re^3 [5], ROLO, [16] and Siamese-FCNN [2]. I tried all these models in order to compare them and find the fastest one.

I assume all tracking models accuracy is enough for my problem due to I only need the UAV to follow simple movements so I mostly centered these experiments on finding a fast model that can fit to my needs. For comparing these models I did some experimentation over them. Using a dataset of a short video sequence I considered good for the experiments and checked the capability of each tracker over that sequence. The experimentation done on each model consists in getting the pretrained model provided by the author, developing myself the rest of the pipeline of the tracker and testing it with the video sequence measuring the accuracy and the time it takes in my machine in order to compare them in the same conditions. For measuring the accuracy of each model I used IOU (*Intersection Over Union*) metrics and all the models are run on CPU. Each model is tested over two experiments. One which tests the model where the input is always the correct position where the object is as we can test properly the accuracy of the tracker assuming the object has not been lost and another experiment where the input location is the output of the model in the previous frame to test the

robustness of the model. The video sequence chosen for the experiments consists in a shot of a man handling an small ball moving it around the screen and throwing it to the air quickly several times so it can be used to test high speeds and speed changes on the target. Also, the ball changes its color when its suspended in the air so its also proper for testing the tracker over appearance changes.

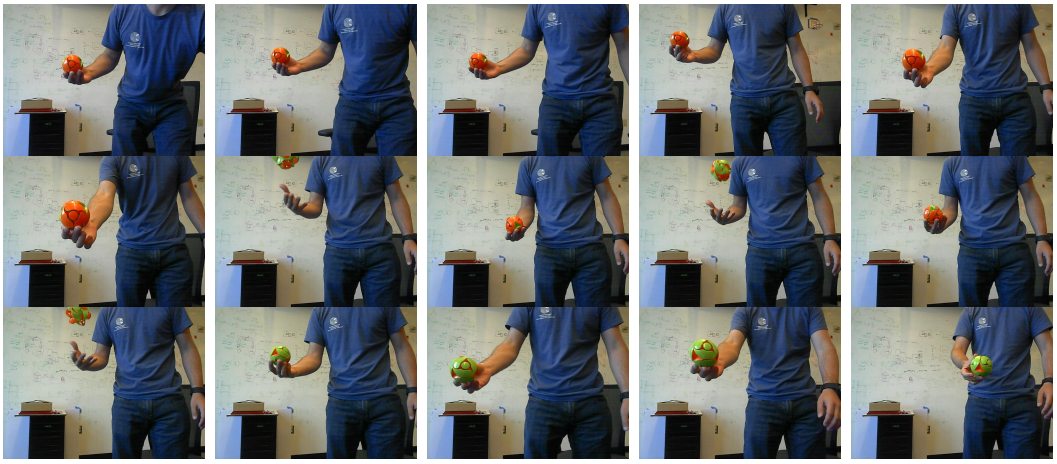


Figure 5: Sample of the video sequence used in the experiments.

3.1 GOTURN

One of the models I tested is GOTURN [8] (Generic Object Tracking Using Regression Networks). GOTURN was presented highlighting its speed, being one of the few deep learning trackers able to run at real time by this time so, according to its paper, it runs at 100 fps.

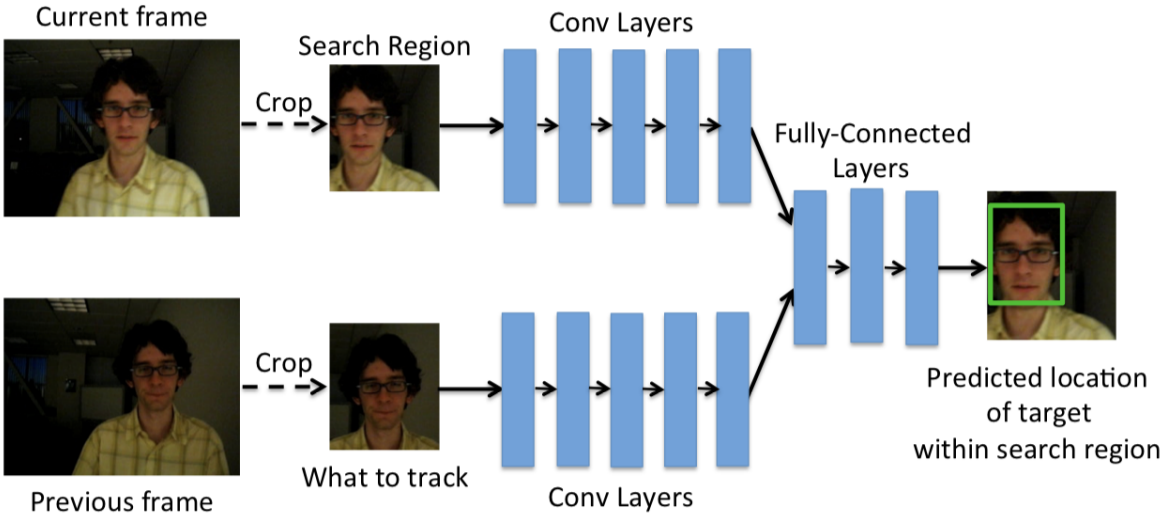


Figure 6: GOTURN tracker architecture.

This tracker receives two inputs, each one corresponding to a different frame and gives a box region as output represented with four coordinates x, y, $width$, $height$ of values from 0 to 1, representing the normalized value of the position on the input image. Each input is a crop of a same region of the previous and the current frame. This crop is determined by the current bounding box of the object with an added padding of half of the width or the height in left and right and in up and bottom respectively. Both inputs are resized to 227×227.

The network consists in two branches of convolutional layers corresponding to the AlexNet's [13] convolutional layers which are concatenated and followed by three fully-connected layers of 4096 neurons with ReLU activations and a final 4 nodes FC layer, each one representing a coordinate.

Figure 7: GOTURN training samples.

The training process is done by a set consisting of a collection of videos in which a subset of frames in each video are labeled with the location of some object. For each successive pair of frames in the training set, the frames are cropped as described before. During training time, this pair of frames are fed into the network and are attempted to predict how the object has moved from the first frame to the second frame. The dataset is also augmented by feeding the network with a same frame, shifting the searching region crop in order to determine its position.

3.1.1 Experimentation

The testing code for the tracker can be found at Appendix A.1 on page 50. The code is very simple: for each frame, I crop it and the previous frame. The crop region is determined by the location of the object in the previous frame and a pad to this region,

in this case I use a pad of 50% of the dimensions of the box that encloses the object as the author proves it is appropriate for the task. Then I use these crops as the input of the model. From the model we receive as output the location of the object in the actual frame. This location is represented relative to the input images (the crops) so I had to transform them in order to be relative to the frame. For the continuous test I store the location of the predicted bounding box and the next crops are done based on this location.

The results I got with this tracker are the following:

	Frame rate	IOU	IOU_{cont}
GOTURN	16.67 fps	0.92	0.54

Table 1: Table of results of GOTURN tracker.

The tracker could track correctly all the sequence without getting lost despite there were some little confusions on which the tracker confused the ball with the hand it was handling it. I got an IOU of 0.92 in the single frame experiment and 0.54 in the continuous experiment. We can see how the accuracy gets worse in the continuous experiment due to the small fails of the tracker such as the confusions commented before and the size of the predicted region, that most times was bigger (but enclosing well the target) than the ground truth.

3.2 Re^3

Re^3 [5] (Real-Time Recurrent Regression Networks for Visual Tracking of Generic Objects) is a more recent and advanced network supposed to work properly at real time. It is also a generic object tracker so in principle fits on my needs. According to the paper, it is able to run at 150 fps on an Nvidia Titan X (Pascal).

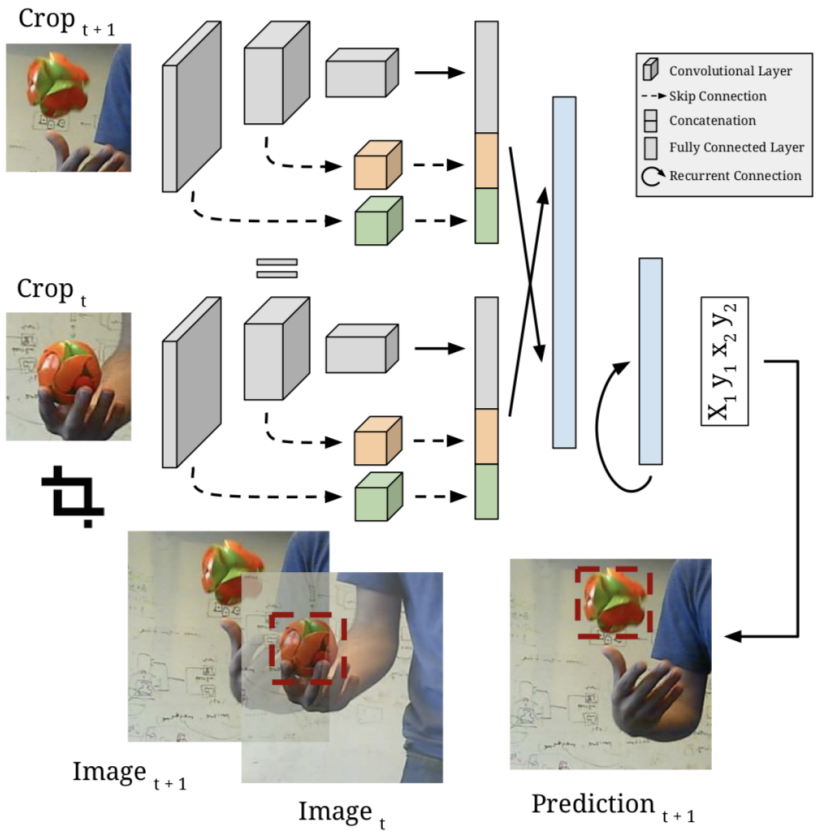


Figure 8: Re^3 tracking pipeline.

Its tracking pipeline (see figure 8) is very similar to GOTURN. It receives two image inputs, each one a crop of a region of an image, one of the current frame and another of the last frame, and predicts 4 values corresponding to the bounding box of the object that is being tracked x_1, y_1, x_2, y_2.

When the two images are received, they are processed by a convolutional network to get its visual features. After that two recurrent LSTM layers are used to retain the information of the appearance of what is being tracked and the path that is has been following, fact that prevents many occlusion errors during tracking. Finally, there is a regression layer to determine the coordinates of the object.

The convolutional part of the network consists in, as GOTURN, an AlexNet architecture [13]. The two images are processed by the same convolutional layers in a batch of two samples. On the end of the convolutional layers, the output is flattened (including the batch dimension) and are concatenated to two skip connections of the convolutional layers. These connections are done on the first and mid layers. This is done due to each part of the network is centered on detecting different features of the

images [23]. For example, it has been demonstrated that edges and corners are detected on the first layers of the network and other more advanced features like shapes are detected on the intermediate layers [23]. All these features can be useful to determine the position of the object we are following as we could compare all these low and high level features among both frames. This concatenation is connected directly to a stack of two LSTM that are responsible of remembering the appearance and the motion information of the target. This part is very helpful to prevent losing the object during occlusions. Then, on the top of the network there is a fully connected layer with four outputs, acting as a regression layer that determines the location of the bounding box that encloses the object.

The network has been trained using a combination of real and synthetic data. This has resulted in a tracker being able to work on large variety of object types, allowing it to recognize not only the objects it has been trained for but any object in the scene. For training on real videos, two large datasets were used: the training set of ILSVRC 2016 Object detection from Video dataset (Imagenet video) [18] and the Amsterdam Library of Ordinary Videos 300+ (ALOV) [20]. On the other hand, the generated synthetic data includes random samples of images of an object randomly patched to simulate occlusions, heavy movements, target size variations or aspect ratio modifications with Gaussian noise.

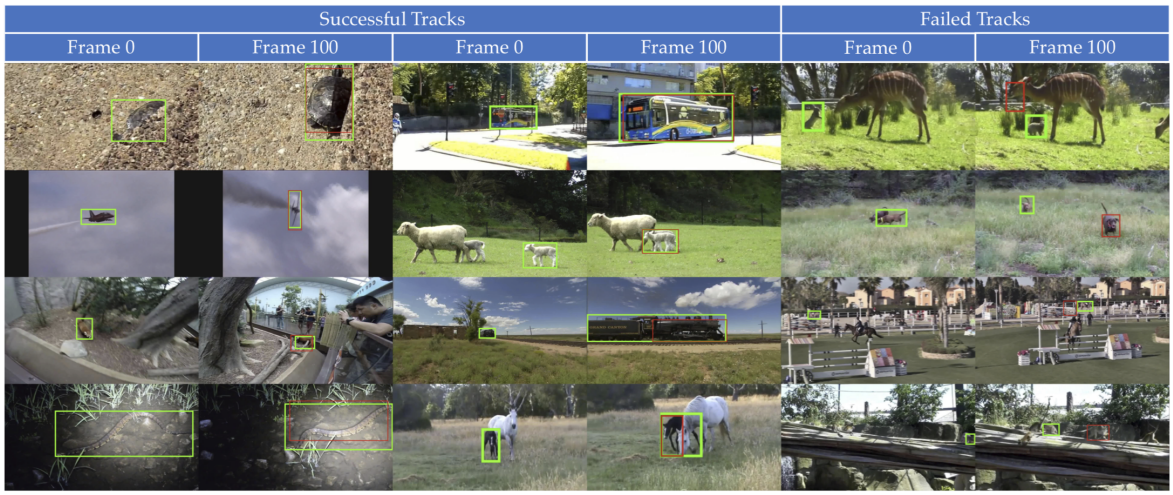


Figure 9: Re^3 prediction samples.

The tracker has been tested on famous tracking benchmarks as VOT 2014 [12] and 2016 [10] test suites as well as on Imagenet Video [18] validation set on which lots

of scenarios such as large appearance change, heavy occlusions, and camera motion are tested. On Figure 9 on the facing page we can appreciate the quality of the tracker after 100 frames, being able to determine the position despite size changes, shape deformations, occlusions or the presence of visually similar objects on the image. However, we have failed tracking examples, most of them due to difficult situations like a very small size of the target or the previously mentioned.

Moreover, according to the paper [5], the tracker has been tested against challenging "never-before-seen" (quoted from the paper) domains in order to gauge Re^3's effectiveness on robotic applications, tested on shaky cellphone cameras, car dashcam footage of moving target and UAV footage of multiple people simultaneously so this tracker is expected to properly carry out the task.

3.2.1 Experimentation

The code of the experimentation done over this tracker can be found on Appendix B.1 on page 53. The code of the experiment is pretty much the same than GOTURN 3.1 on page 18 code. For each frame, we take a crop of that and the previous frame of the region that represents the bounding box that encloses the target in the previous frame with an added padding (in this case is the same as in GOTURN, 50% of the dimensions). This crops are resized to 227×227 pixels in order to fit in the model. I feed the model with these crops as inputs and I get as output the current position of the target. The difference with GOTURN is that we have to feed the network not only with the images but also with the initial state of the LSTM so in each iteration, we store the new state of the LSTM cell and in the next one we feed the network with it. After that, the output received from the network is, as in GOTURN, relative to the crops so I need to transform them to be relative to the full frame.

Moreover, to ensure the robustness of the tracker and taking the advantage of the *remembering* feature of LSTMs, I store the output state of the LSTM cell in the first iteration in order to remember the original features from the target. Each n iterations (in this case 32 as the author demonstrated it worked well) we feed the network with this *original* state instead of the previous one to avoid looses in case the network is not recognizing well the features of the target.

	Frame rate	IOU	IOU_{cont}
Re^3	12.50 fps	0.87	0.76

Table 2: Table of results of Re^3 tracker.

This tracker gave very good performance results in terms of accuracy and robustness. When testing it on the video sequence in the continuous experiment it followed very accurately the target without loosing it at all. In this case there isn't any visible fail on predictions apart from little displacements of the predicted bounding box in respect of the ground truth. Despite we have a worse IOU on the single experiments than GOTURN (0.87 front 0.92), the tracker demonstrates more robustness on the continuous tracking, getting the result of 0.76, much better than GOTURN's 0.54.

3.3 ROLO

ROLO [16] is a powerful tracker that proposed an approach of recurrent convolutional neural network. This tracker use recurrent layers to memorize the object that is being tracked, the trajectory it is following and similar features that helps the network to determine the location of the target.

The tracker's name comes from *Recurrent YOLO* and it is because its highlight is that uses as base a famous object detector, *YOLO* [17], as a base net to detect the location of all the objects in the image.

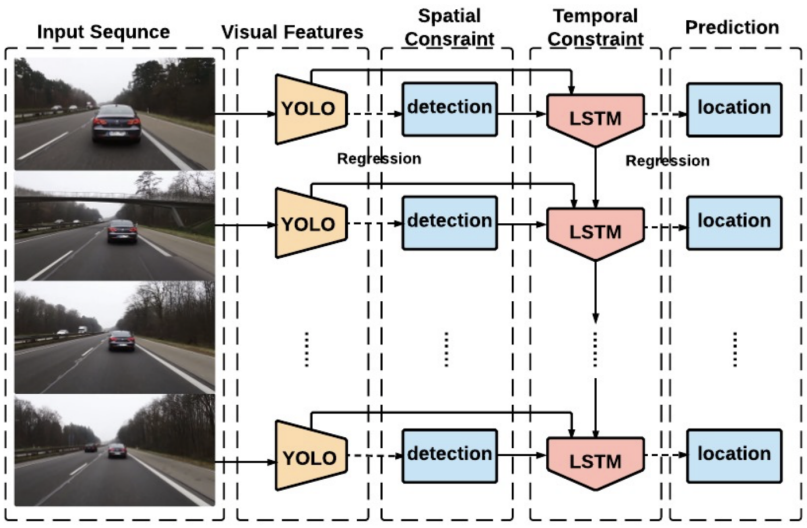


Figure 10: How ROLO works.

Basically, the network pipeline is, given a frame sequence and the bounding box of the desired target object in the previous frame (in the first frame it has to be initialized), the YOLO network [17] detects the object in the image, this output feeds the LSTM layers that remembers the features of the target and a final regression layer determines the location of the target object in the image.

3.3.1 YOLO

YOLO [17] (*You only look once*) was presented as a new approach on object detection techniques. Its main feature is that it determines the object position and classifies it in a single network, processing the image a single time, that is why its name is *You only look once*, in contrast to other object detection models like slicing window approach.

To do this, YOLO produces an output tensor of $S \times S \times (B \times 5 + C)$. This tensor represents a grid on the image of size $S \times S$ on which each grid element is responsible to determine the position of B bounding boxes and the class probabilities of C classes. Each grid element has a size of $B \times 5 + C$. The first $B \times 5$ elements of each cell represents the coordinates and the confidence of the B possible bounding boxes that are on that space in the image. The remaining C elements correspond to the probabilities of each class on the dataset appearing on that space.

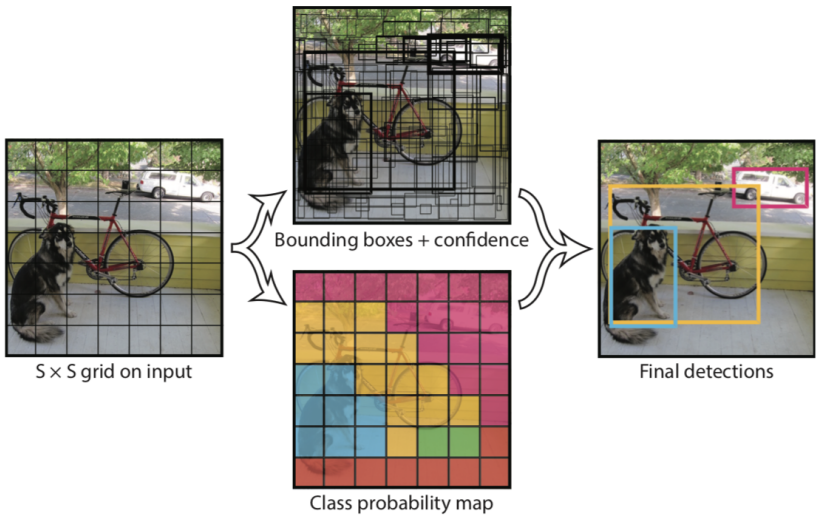


Figure 11: YOLO sample output.

YOLO is designed to be evaluated on Pascal VOC detection dataset [4]. Its convolutional layers' architecture is a version of GoogLeNet [21] architecture with some

modification. It has 24 convolutional layers followed by 2 fully-connected layers and instead the inception modules of GoogLeNet, it simply has 1×1 reduction layers followed by 3×3 convolutional layers. For the output of the network they use a grid of 7×7 ($S = 7$) and predicts as much as 2 boxes per grid cell. Also, as Pascal VOC has 20 classes, the output tensor has a shape of $7 \times 7 \times (2 \times 5 + 20) = 7 \times 7 \times 30$. In Figure 11 on the previous page there is an example of a YOLO prediction. First there is the input image on a grid. Then, on the top we can see the bounding boxes detected by each grid cell and its confidence (represented by the stroke strength). Below we can see the classification of each cell. Finally, after processing these data, as output we have the labeled bounding box of each detected element of the image.

Apart from that architecture (called *full YOLO*), YOLO has more versions of the network including one called *fast YOLO* [19]. That version, as the name says, does faster predictions, being up to three times faster than the full YOLO. This architecture focus on speed in exchange of loosing accuracy on predictions. However, as the aim of ROLO is to be as fast as possible, it uses fast YOLO as backend.

In terms of training, YOLO's convolutional layers are pretrained on ImageNet dataset [3] in order to detect image features. Then the full network is trained with Pascal VOC 2007 and 2012 [4].

3.3.2 LSTM layer

For each prediction, the input of the LSTM layer is a combination of the features representation from the convolutional layers and the detection information from the fully connected layers of the last n frame outputs from YOLO. As output we have 4 values x, y, w, h representing the bounding box center expressed in a value relative to the image dimensions by x and y and the width and height of the bounding box by w and h respectively, also relative to the image dimensions.

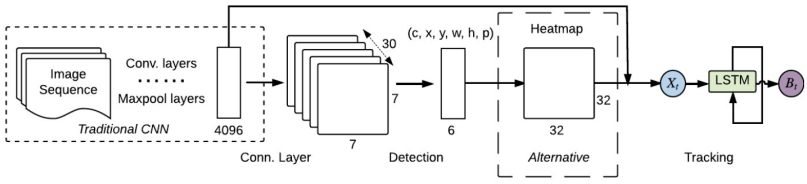


Figure 12: Full ROLO architecture

3.3.3 Experimentation

The code of the experiments I did on this tracker can be found at Appendix C.1 on page 75. In contrast to the previous trackers, this network does not take two image inputs, however, it takes the current frame and the previous location of the object represented as a bounding box. The pipeline is a bit more complex than GOTURN and Re^3 but it is still quite simple. First of all, we take the output boxes of the YOLO network, just the boxes as we do not need to know the label of each detection. We take among all these detections the most suitable to be our target. It is done by computing the IOU of each detection over the location of the target. After that, we feed the recurrent network with a concatenation of the output of one of the last layers of the YOLO network (as they represent the detected features of the image) and the location of the most suitable location found by YOLO. Apart from that, on each iteration I store the output state of the LSTM cells and I feed them with it on the next iteration for preserving the moving features of the target.

	Frame rate	IOU	IOU_{cont}
ROLO	1.96 fps	0.09	0.06

Table 3: Table of results of ROLO tracker.

In terms of accuracy these results are by far worse than the other tested trackers. It is because as the recurrent network only interprets the output of YOLO, the tracker is only capable of tracking objects that are detected by YOLO. As this YOLO version does not track balls, the results I take are very poor. Maybe it's not the best experiment for benchmarking the capabilities of the tracker but it is enough to demonstrate it does not fit in my needs as I need an arbitrary object tracker in order to track any possible target. Moreover, the frame rate at which it runs is fairly lower than the other.

3.4 Siamese FCNN

The other model I tested is presented in the paper *Full-Convolutional Siamese Networks for Object Tracking* [2]. It consists in a fully-convolutional architecture. This tracker centers on being as simple as possible, trying to demonstrate the capabilities of fully convolutional networks on computer vision problems, able to detect very well visual

features on images. As the network is try

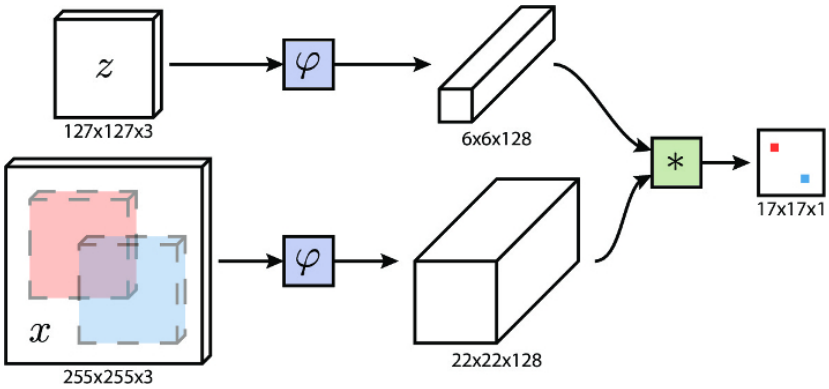


Figure 13: Siamese Fully Convolutional tracker architecture.

The tracking pipeline of the model is quite simple. The network receives two image inputs in a very similar way to GOTURN (see Section 3.1 on page 18) or Re^3 (see Section 3.2 on page 20). One of those inputs corresponds to the target that is being followed. That image is the crop of the last frame where the target is located. The other image is a search region of the new frame. This region is the one that was the target in the previous frame with a padding to prevent heavy movements. These two inputs are processed independently in a siamese structure and concatenated near the top of the model. After that, the network predicts an score probability map of presence of the object on the image. The probability map represents a portion of the search region. In Figure 13 we can see an example of how this works. On the output of the network are two pixels show. Each pixel is responsible of predicting the probability of the target being in a certain position of the searching image, region that we can see on the x input on the diagram. Thus, if the red pixel probabilities are the highest, the target is located on the red region on the x input. Besides, if more than one score are high, the bounding box of the location of the target is the one that encloses the regions than represents that two scores. On Figure 14 on the facing page we can see an example of both inputs of the network and its output. Above there is the target image and below, the searching image with the output bounding box.

Figure 14: Siamese-FCNN sample.

On each evaluation three different search regions are passed to the network. All having the same center, each one have an scale in order to prevent scale ratio changes on the target. These images are evaluated by the network in a single forward-pass assembled in a batch of scaled images. Then, the resulting feature maps are all combined using cross-correlation. On the final design of the tracker, the network is fed with 3 scales as it was tested [2] to offer good results without affecting too much to speed.

The network was trained on Imagenet Video dataset [18] and tested comparing itself with VOT 2014 [12] best 10 trackers and VOT 15 [11] best 40 participants, reaching the state of art results.

3.4.1 Experimentation

The code of the experiments done on this network can be found on Appendix D.1 on page 77. As a great part of the pipeline of the tracker is implemented on TensorFlow in the network itself I did not have to write many code, however, I will explain the way it works. The tracker has several inputs: one image that is responsible of determining the *target* that it has to look for and n images of the possible *search regions* (see Figure 13 on the facing page). The *target* has a size of 127×127 and it is a crop of the bounding box that encloses the target in the previous frame. The other n images correspond to the search regions where the target can be in the current frame. These search regions are in the same position as the target was in the previous frame, each one with a different size for making the tracker capable of predicting the position despite size changes. In this case, I used the pretrained tracker with $n = 3$ as the author demonstrated it has a good relation speed-accuracy [2].

	Frame rate	IOU	IOU_{cont}
Siamese-FC	1.47 fps	0.63	0.12

Table 4: Table of results of Siamese-FC tracker.

Despite the author showed very good results with that tracker, I did not get such as good results. On the individual experimentation I got an IOU of 0.63. This can be due to the tracker is not capable of provide a totally accurate result, as said in section 3.4 on page 27. Although they are not as good enough for determining if the tracker is good enough for my problem, when testing the continuous experiment on it, the tracker lost the target very easily when it was moving a bit quick so it hardly work on the UAV.

4 Adapting the net to the device

The next step on the project was to implement the network on a mobile platform in order to build an application that interacts with the UAV and controls it. The objective is to find the most suitable network for the task, convert the model to a compatible format that could be used on a deep learning mobile framework and coding all the pipeline of the tracker.

4.1 Choosing the best network

Once I understood and analyzed some of the most relevant state of art tracking networks I had to proceed to find the most appropriate for my project. All four networks demonstrated on benchmark to work really well in terms of accuracy, however, on my experiments done specifically keeping the problem in mind, not all of them had satisfactory results so I had to find a robust enough tracker running as fast as possible.

As it is not computationally possible to process every frame obtained by the camera, in this case *real time* means that the transition between frames is as smooth as possible, minimizing the variation of the current frame with respect to the previous one. This smoothness can be obtained with a frame rate pf 15 fps approximately for the human eye so I looked for getting close or overcoming that number.

	Frame rate	IOU	IOU_{cont}
GOTURN	**16.67 fps**	0.92	0.54
Re^3	12.50 fps	0.87	**0.76**
ROLO	1.96 fps	0.09	0.06
Siamese-fc	1.47 fps	0.63	0.12

Table 5: Table of test results.

The most relevant information on table 5 is the frame rate and IOU_{cont} that represents the speed at which they run and On this test we can conclude the most suitable models for the project are GOTURN and Re^3. Those frameworks in fact, have a very similar architecture, having some convolutional layers in order to detect visual features connected to a deep regressor. The main difference is that Re^3 uses a couple of LSTM layers to retain certain information about the target object.

4.2 Mobile deep leaning frameworks

Other important aspect to keep in mind when implementing this kind of model on mobile platforms is the framework that is being used. I needed a deep learning framework for mobile devices that could work as fast as possible despite the low capabilities of these platforms.

Figure 15: Mobile deep learning frameworks.

As deep learning is becoming more and more popular in last few years and once it has proved all the advantages it can bring to our lives, the need of using deep learning on smartphones is becoming bigger. Because of this, are emerging mobile deep learning frameworks for solving this. Despite most of them are very new and are in prematures versions (the following frameworks were published on the last year), they are very powerful and offer running deep learning models at the highest speed a mobile device allows. I made an analysis of three of the most important and famous mobile deep learning frameworks in order to checkout which one fits best on the project.

4.2.1 TensorFlow lite

TensorFlow is one of the most powerful deep learning libraries right now. It was firstly a private library developed by Google for its internal use before its source was published.

This library is designed to be a high performance computational library. It is based on operations on multidimensional arrays called tensors structured in a computational graph. This architecture allows easy deployment across a variety of platforms, including

CPU and GPU executions, and a great scalability on many environments. This architecture also is very useful for neural networks which can be expressed as computational graphs.

TensorFlow is originally developed for Unix environments, however, it has a lite version that is available also for other platforms like Android or iOS. TensorFlow lite is a version of TensorFlow released on late 2017 as an evolution of TensorFlow Mobile that allows running pretrained TensorFlow models on mobile devices. It is a simplified version of the library that supports a subset of the operands of the full version with some other capabilities.

This library allows running very efficiently deep learning models on mobile devices. However, due to the capabilities of these devices, the operations work too slow making difficult and limiting the use of some models in certain situations like in this case, real time object tracking. For solving this, TensorFlow lite allows us to run a simplified version of the model with quantized weights. This means that, instead of storing the weights in a typical 32 bits precision floating point, they are stored using just 8 bits and performing all the computation with that precision. Quantization is done by constraining the values of a set of floating point number in order to be able to represent them in a discrete set format. This allows performing all the computation using 8 bits integers, resulting in a performance up to 4 times faster.

For using a model on a mobile device we have first to convert it to the lite format, TensorFlow provides many tools on his official GitHub repo [2] for helping us to it: an easy graph editor for deleting nodes used just in training time, a converter to lite format, a quantizer... But since many operators of the full version of TensorFlow are not supported by the lite version, we have to design the network in advance in order to satisfy that requirement otherwise we won't be able to convert the model.

4.2.2 Caffe2

Original version of Caffe was well known by its high performance. It was developed by Berkelei AI Research (BAIR) [3] in a very optimized way and it's commonly used for deploying models that needs high speed like computer vision models that need to process image videos at real time.

[2] https://github.com/tensorflow/tensorflow

[3] http://bair.berkeley.edu

Caffe2 was developed as an improved version of Caffe trying to innovate some aspects on deep learning frameworks. It is being developed by Facebook and has C++, Python, Android and iOS API's, in contrast to Caffe that only runs on C++. As the previous version, its intention is to develop a fast as possible deep learning framework. This version highlights also by its scalability, as one of its slogans says *Code once, run anywhere*. Its aim is to develop a framework that could run models on any application you want, from backend high performance servers to smartphones or embedded devices with low capabilities.

In contrast to TensorFlow lite, this framework is totally cross-platform so there is no need to convert models or adjust them to fit in the mobile version as any model coded on Caffe2 runs perfectly on every device that runs Caffe2. Also, Caffe2 uses graphical acceleration on mobile devices (Metal on iOS and ARM graphic instructions on Android) highly improving the performance of floating point computations.

4.2.3 CoreML

CoreML is a Deep Learning framework developed by Apple for their own devices which was presented on WWDC 2017 event. It is oriented specifically for mobile devices in order to make deep learning being closer to users.

CoreML makes predictions of models previously trained in other frameworks. You only have to train a model or take a pretrained one and convert it to CoreML format using coremltools. Coremltools is a python package that Apple provides to convert your models. Coremltools now has converter for Keras, Caffe, Scikit-Learn, XGBoost and LIBSVM. Moreover there are several unofficial converters for using other format models like tf-coreml [4] for TensorFlow models.

This framework is designed focusing on high performance. For this, is developed using low level high performance primitives of iOS like Accelerate and BNNS libraries, as well as Metal Performance Shaders which uses the device's GPU for high speed computations.

[4]https://github.com/tf-coreml/tf-coreml

4.3 First approach: Re^3

As Re^3 gave me the best accuracy with one of the best frame rates, I first tried to implement this model on the mobile device for testing it.

4.3.1 CoreML

For this approach I first used CoreML, as the most powerful mobile device I had available for using was an iPhone 8.

In principle, the use of a model in CoreML is quite simple. I first have to implement the model in a deep learning framework which has to be compatible with the CoreML converters (Keras, Caffe...). As I've worked before with Keras and I feel comfortable with it, I decided to use Keras for the task. There were, however, some aspects to take into account:

- The model written by the author takes 2 image inputs and these inputs are given in a batch of size 2. CoreML does not support multiple samples in a batch so what I did was to flatten the inputs and then *unflatten* them. Instead of having an input of size $(2, 227, 227, 3)$, my input was of size $(1, 227 \times 227 \times 2, 3)$.

- The tracker uses some grouped convolutions. That kind of convolutions are not supported natively in Keras so I had to implement them.

- There were some operations on TensorFlow that did not have a direct equivalent in Keras. As a result, I had to implement some custom layers on Keras to replace them.

For the solving the input problem I had to reshape the input tensor to the original dimensions $(2, 227, 227, 3)$. Keras has a native Reshape layer but it does not support batch channel reshaping so I used a lambda layer that reshaped the input tensor with the low level keras backend.

On the other hand, the grouped convolution can be easily implemented in Keras. This kind of convolution consists on dividing the input in n groups on the batch channel and processing them separately, each split convolving with a different weight tensor. For doing that I just added an split layer before each group convolution, I passed each

of them to a different convolution layer and finally I concatenated the outputs on the batch channel.

Then, I had to implement two custom layers that were not implemented on Keras:

- *Local Response Normalization* layer. The LRN layer was implemented once in Keras but it is not available anymore. This type of normalization was very used in the past but when Batch Normalization arrived, it became more popular due to its advantages front other normalization techniques (for example, allowing build deeper networks) and Local Response Normalization was less and less used. The implementation of LRN on keras can be found on Appendix B.2.1 on page 55.

- *LSTM* layer. Keras has its own implementation of an LSTM layer, however, the tracker was originally developed on Caffe and the implementation that has Caffe of the LSTM layer is slightly different from the keras version. As a result, I had to implement the exact version of Caffe's LSTM on Keras. The implementation of this layer can be found on Appendix B.2.1 on page 55.

Once I had all the network implemented (see Appendix B.2.2 on page 57) I proceeded to importing the TensorFlow weights to the keras model. For easily doing this, I exported the weights in a numpy format and then I imported them to keras. This code can be found on Appendix B.2.3 on page 58. I simply load the weigths of each layer and assign them to its corresponding layer in keras. After that I could finally test the model. For testing it I just coded all the pipeline and tested the tracker visually over a video sequence (see the code of this experiment in Appendix B.2.4 on page 60).

At this point that I had the tracker running correctly on Keras I just have to use the converter from the coremltools provided by CoreML for converting the model to the CoreML format. The way the CoreML converter works is very simple, you just setup basic configuration of the model for CoreML such as the name of inputs and outputs, a little description... And also specify the interface of the custom layers setting up the weights of each layer, too. The code I wrote for this is very straightforward and can be found on Appendix B.2.5 on page 62.

After that, when having the model in CoreML format I still have to write the code of each custom layer for CoreML. CoreML offers the possibility of using custom Keras layers if you write the equivalent code in CoreML so I had to write the custom layers I

used in Swift. Apart from the previously mentioned (the LSTM and LRN layers) I used some lambda layers for splitting, reshaping and concatenating as Keras did not allow me to do exactly what I wanted with its native layers (Keras restrict modifications on the batch channel). For developing these layers I had to take some aspects into account:

- All CoreML tensors have 5 dimensions (S, B, C, H, W) where $S = Sequence$, $B = Batch$, $C = Channel$, $H = Height$ and $W = Width$ so any operation on them cannot add or remove any dimensions.

- Due to the dimension ordering, CoreML is a Channel First framework. This means that the channel dimension comes before height and width dimensions, in contrast to Keras that is Channel Last, having the last three channels in order (H, W, B) instead of (B, H, W) so I have to take this into account when iterating over tensors.

- The channel ordering is different in CoreML than in Keras. While in Keras is RGB in CoreML is BGR. This actually does not matter in my model because it only affects when treating directly with images and I do not do significant changes on the first layer.

Once I knew the CoreML basics I passed to develop the custom layers. Custom layers code can be seen on Appendix B.2.6 on page 64. Apart from all the layers I needed for the model, I wrote a debugging layer that simply acts as a log for the network printing the output of each layer (see Appendix B.2.6.6 on page 73). For developing these layers I used the math native library from Swift and the Accelerate library that provides high performance computations over matrices.

4.3.1.1 Problematics

During the developing of the model on CoreML I faced primarily two problematics that made it a bit difficult and even impossible to develop the model on this framework. These problematics are the following.

Coremltools bug

Once I debugged each custom layer separately and tested that they were working correctly I tried a forward pass of the full network. For this, I took a pair of images and an initial state of a zero vector for the LSTMs and passed them to the networks in Keras and in CoreML for checking out the output in both models. Surprisingly, given the same inputs in Keras and in CoreML, the output was not even similar. Using the debugging layer (see Print layer in Appendix B.2.6.6 on page 73) I realized that at the end of all convolutional layers there is a concatenation of the last layer with 4 residual connections. The output of this concatenation was incorrect. Going deeper on this I found that these residual connections had a permutation (as the network was trained on Caffe originally, when flattening the convolutional layers they have to be permuted to fit the Channel First dimension ordering of Caffe) and it turns that this permutation should have been nullified in CoreML as the dimension ordering is the same as in Caffe, however, it was performed incorrectly. For solving this, I took the nearest custom layer, concretely the FlattenBatch layer that is just after the permutation, and solve this permutation manually (see Appendix B.2.6.3). After this, the output of this layer was the same as in Keras.

CoreML floating point precision

Despite having solved the problem and getting correct results at this point of the network, I still had bad results on the final output. Checking the outputs of each layer separately I found out that there was an small error on the output of each layer that might be negligible (near 10^{-8} in the first layers). However, that error was bigger and bigger as the network got deeper. After all the convolutional layers, near the end of the network, the error was also practically negligible (near 10^{-5}) but when passing through the layers which has a great number of parameters, both LSTMs and two Fully Connected layers, the error was so big that the result was completely different. I concluded that this might happen due to the floating point precision on common operations of the Accelerate library that is visible on very deep networks like that.

I checked it out why it did not happen on other pretrained models that can be found on the Internet. I saw that, while most models weights a couple of hundreds of megabytes, my model weighted near 1GB, having many more weight parameters than

most models. I tested the VGG net implementation imported from Keras, which was the heavier model I found (near 500MB). I found that the error was present on this network as well but it was not big enough for corrupting the result so the network worked completely well. However, as my model was bigger, the final output of my model was incorrect.

4.3.2 TensorFlow lite

As a result of this failed conversion, I considered using the model on Android. I checked out the frameworks mentioned on Sections 4.2.1 on page 32 and 4.2.2 on page 33. I didn't mind choosing one or another framework so I did a little experiment for checking out which was the fastest one. I downloaded two pretrained models, one of Caffe2 and one of tflite and measured the frame rate it could work on my Android device. I also took the same models in Keras to test its speed on my computer as I can compare them. For tflite I chose an Inception v3 [22] and for Caffe2 I tried a ResNet50 [7]. In tflite, I also tried two versions of Inception v3, one with quantized weights and other without quantization.

	Keras	Android	Speedup
ResNet50 (Caffe2)	15.1 fps	3.7 fps	0.24
Inception v3 (tflite)	20.4 fps	10.1 fps	0,49
Inception v3 (tflite quantized)	20.4 fps	64.2 fps	**3.14**

Table 6: Results of tflite-Caffe2 comparation.

Given these results my first choice for implementing the model was TensorFlow lite. I needed of course the highest speed as possible as the model had to run at real time. Having the speedups as reference, Re^3 would work at less than 4 fps on Caffe2, at 6 fps on TensorFlow lite without quantization and at more than 40 fps with quantization. 4 and 6 fps would not be enough for a tracker for running at real time so the only remaining possibility was TensorFlow with quantization.

The process of converting a TensorFlow model to a TensorFlow lite format is very straightforward. First I had to *freeze* the TensorFlow graph of the session. A frozen TensorFlow graph is the way TensorFlow allows to save all the computational graph and its weights in a single file in order to being able to restore it easily afterwards. For

freezing the graph you first have to save it in a *protobuf* format. This can be done easily calling a function on TensorFlow:

```
tf.train.write_graph(sess.graph_def, dir, file + "pbtxt")
```

Being *sess* the session where the graph is located, *dir* the directory where the frozen graph is going to be stored and *file* the output filename without extension. After that, we have to call a script that provides TensorFlow in its source code:

```
python ~/tensorflow/tensorflow/python/tools/freeze_graph.py \
    --input_graph logs/re3.pbtxt \
    --input_binary false \
    --input_checkpoint logs/re3.ckpt  \
    --output_graph logs/re3_frozen.pb \
    --output_node_names \
        're3/fc_output/add' \
        're3/lstm1/rnn/LSTM/forget_gate/new_state' \
        're3/lstm1/rnn/LSTM/output_gate/cell_output' \
        're3/lstm2/rnn/LSTM/forget_gate/new_state' \
        're3/lstm2/rnn/LSTM/output_gate/cell_output'
```

where:

- *freeze_graph.py* is the python script provided by TensorFlow that gets the protobuf graph and the checkpoint with the weights of the model and *freezes* it to a single file.
- *input_graph* is the graph in protobuf format.
- *input_binary* tells if the graph is stored in binary o in text format. In this case, *false* means it is text format.
- *input_checkpoint* is the path to the model checkpoint containing all the weights of the model.
- *output_graph* is the output file where the frozen graph is going to be stored.
- *output_node_names* is the list of output nodes in thee graph. In this case they are the output tensor that of the bounding box enclosing the target and the new states of the LSTM cells.

After having the frozen graph ready, I proceeded to converting it to the TensorFlow lite format. As well as for freezing the graph, TensorFlow provides a powerful tool for converting the graph to the lite format. The script called *toco* includes a feature for quantizing the weights of the graph, which I needed. That tool is also very straightforward and can be used with a simple bash command:

```
bazel-bin/tensorflow/contrib/lite/toco/toco -- \
    --input_file=/Users/enrique/workspace/tfg/tracking/re3-tensorflow/logs/
                                        re3_frozen.pb \
    --input_format=TENSORFLOW_GRAPHDEF \
    --output_format=TFLITE \
    --output_file=/Users/enrique/workspace/tfg/tracking/re3-tensorflow/logs/
                                        re3_quantized.tflite \
    --inference_type=QUANTIZED_UINT8 \
    --input_arrays=
        Placeholder,
        Placeholder_1,
        Placeholder_2,
        Placeholder_3,
        Placeholder_4 \
    --output_arrays=
        're3/fc_output/add',
        're3/lstm1/rnn/LSTM/forget_gate/new_state',
        're3/lstm1/rnn/LSTM/output_gate/cell_output',
        're3/lstm2/rnn/LSTM/forget_gate/new_state',
        're3/lstm2/rnn/LSTM/output_gate/cell_output' \
    --input_shapes=2,227,227,3:1,1024:1,1024:1,1024:1,1024
```

Being:

- *toco* the tool name.
- *input_file* the path of the frozen graph.
- *input_format* the format of the graph. In this case a simple TensorFlow graph definition.
- *output_format* the output desired format. In this case a TensorFlow lite graph.
- *output_file* the path where the new graph is going to be saved.
- *inference_type* the type of the inference. This can be 32 bits float or quantized unsigned 8bit integer. This flag determines if the weights are converted or not to a quantized format. Quantization works as explained in Section 4.2.1 on page 32.
- *input_arrays* the input node names of the graph. They are the placeholders which we feed when doing a forward pass on the network.
- *output_arrays* the output nodes in the graph, the same as when freezing the graph.
- *input_shapes* the shape of each one of the inputs.

This command should have generated the TensorFlow lite model, however, I got an error which said that the LRN operation was not supported bu the quantizer tool

yet and the model could not be converted. I tried to replace the Local Response Normalization operation implementing the layer with other low level operations but they were not supported too. This made me impossible using this model on Android as TensorFlow lite with quantization was the only way I could make the model work at real time on Android.

4.4 Second approach: GOTURN

After having tried all possible alternatives for using Re^3 on a mobile device I finally gave up and decided to use another tracker. The next best tracker I found was GOTURN (see Section 3.1 on page 18) which was also faster than Re^3. This model uses also Local Response Normalization layers so using TensorFlow lite was totally discarded. On the other hand, GOTURN is not as deep as Re^3 so the CoreML option might be feasible.

4.4.1 CoreML

For converting the model to CoreML I decided to use an unofficial converter for converting the model directly the TensorFlow model to the CoreML format. This tool called *tf-coreml* [5] allows converting easily a pretrained model of TensorFlow to CoreML. It works in a very similar way to the *toco* converter of TensorFlow lite. First of all I had to freeze the graph as in Section 4.3.2 on page 39.

First we call the following function with the model graph for saving the graph in a protobuf format.

```
tf.train.write_graph(sess.graph_def, dir, file + "pbtxt")
```

And then we use the TensorFlow tool that freezes the graph:

```
python ~/tensorflow/tensorflow/python/tools/freeze_graph.py \
    --input_graph checkpoints/goturn/goturn_tf.pbtxt \
    --input_binary false \
    --input_checkpoint checkpoints/goturn/goturn.ckpt \
    --output_graph checkpoints/goturn_frozen.pb \
    --output_node_names 'fc4'
```

Where:

- *freeze_graph.py* is the python script provided by TensorFlow that gets the protobuf graph and the checkpoint with the weights of the model and *freezes* it to a single file.

[5] https://github.com/tf-coreml/tf-coreml

- *input_graph* is the graph in protobuf format.
- *input_binary* tells if the graph is stored in binary o in text format. In this case, *false* means it is text format.
- *input_checkpoint* is the path to the model checkpoint containing all the weights of the model.
- *output_graph* is the output file where the frozen graph is going to be stored.
- *output_node_names* is the list of output nodes in thee graph. In this case there is only one output tensor that corresponds to the prediction of bounding box enclosing the target.

Once I had the graph frozen I just had to use the tfcoreml tool. The conversion can be done just with this piece of code:

```
import tfcoreml

input_tensor_shapes = {'image:0': "1, 227, 227, 3",
                       'target:0': "1, 227, 227, 3"}

output_file = "goturn.mlmodel"

output_tensor_names = ["fc4:0"]

frozen_model_file = "checkpoints/goturn_frozen.pb"

coreml_model = tfcoreml.convert(
    tf_model_path=frozen_model_file,
    mlmodel_path=output_file,
    image_input_names=['image:0', 'target:0'],
    input_name_shape_dict=input_tensor_shapes,
    output_feature_names=output_tensor_names)
```

For making the converter work it is only needed to specify the input node names and its shapes and the output node names. We pass these parameters and the frozen graph path to the tool and it automatically generates the CoreML model.

4.4.1.1 Problematics

The conversion of this model was more straightforward than Re^3. However, at this point the model did not work yet. As this converter do only support a restricted sort of TensorFlow operations, I could not use the debugging layer I used when converting from Keras to coreml on Section 4.3.1 on page 35 I could not debug easily the outputs of each layer. As a result, for debugging the network I had to generate several models

with the tfcoreml converter, each of them having a different output tensor so I could check the output of each layer and compare it with the correct output in TensorFlow.

Doing this I found that some permutations are not correctly converted with the tool as they did not take into account the dimension ordering. For fixing this I just modified the tfcoreml code in order to perform the permutations correctly.

4.4.1.2 Results

Finally I could have correctly working the model on CoreML. I realized that the error produced when performing the floating point operations that I had with Re^3 (see Section 4.3.1 on page 35) was still present but was negligible and the output of the network was the same as in TensorFlow.

4.4.1.3 Building the tracker

Once the model was working I started to build all the tracker pipeline in an iOS app. As the native array library in the CoreML package is a bit hard to use, I used a small library called CoreMLHelpers [6] that acts as a wrapper of the MLMultiArray class of CoreML that provides some useful high level functions for managing arrays.

For developing the tracker for iOS, I simply translated the python code to Swift. The code of the tracker can be found on Appendix E.1 on page 79. For making it work you have to initialize first the tracker in order to get the first frame and the location of the target that is going to be traced with the function *startTracking* and then call each iteration the function *track*. This function returns the location of the target in the searching image. The rest of the code works exactly in the same way as the python version wrote for evaluating the model (see Appendix A.1 on page 50).

After doing that I build an small application that gets the images that the quadcopter was taking (as explained in Section 2.2.2 on page 15) and tracks the position of an arbitrary object selected drawing a bounding box on the screen. The tracker worked perfectly with an average of 16 fps.

[6] https://github.com/hollance/CoreMLHelpers

5 Predicting the movement

By the moment I finished the tracker pipeline I had almost everything I needed for making the application work. I only had to generate a movement depending on where the target was located.

For predicting the movement, I considered the simplest scenario where the drone is flying at a high height with the camera looking to the ground and it is following a non-flying target that is located below it. In that case, the target can only move around the ground, reducing all the possible movements of the target to two possible axes, X and Y, being impossible for the target to elevate its position. Thus, the movements that the drone drone has to do to follow the target are just in those axes. We can say that the target is *centered* and the drone has to stand still when the target is located exactly in the center of the image. However, if the target is located on the top of the image, the quadcopter has to go forwards increasing its pitch. In the same way, if the target is not centered on the image horizontally, the quadcopter has to move laterally setting up its roll properly.

For computing accurately the pitch and roll the application sends to the aircraft, I made a simple method that determines it in function of how far is the target from the center of the image:

$$pitch = 2 * (x - 0.5) * maxVel \tag{1}$$

$$roll = 2 * (y - 0.5) * maxVel \tag{2}$$

Where x and y are the coordinates of the position of the center of the target in the image normalized from 0 to 1 (0 the most left/up and 1 the most right/down) and $maxVel$ is a parameter that determines the maximum velocity wanted for the drone to move.

The code of the quadcopter controller can be found on Appendix E.2 on page 81. The main function of the program is the function *update*. It is called continuously and is responsible of the main loop of the controller. Each iteration, I fetch the frame that the camera is getting and if there is a target selected, I use the Tracker class for tracking it. The result of the tracking is passed to the *updateVel* function that is responsible of

computing the pitch and roll that had to be sent to the aircraft and finally, they are sent.

6 Conclusions

As a general conclusion of the project I think the results are very satisfactory. All the main objectives were correctly achieved maybe with better results that I expected:

- *Control a UAV through software.* I learned how to use the DJI SDK for controlling the quadcopter via an application autonomously (see Section 2.1 on page 11)
- *Testing the state of art visual trackers.* I got in the state of art of visual trackers in order to know how to build my system. I found a tracker that fit my needs, being able to follow arbitrary targets and that would be able to run at real time (see Section 3 on page 17)
- *Know about the mobile deep learning frameworks.* I implemented the tracker on a mobile device. For this I had to chose the most appropriate deep learning framework (see Section **??** on page ??) in order to make the model work as better as possible.
- *Port the tracker to a mobile platform.* I implemented the tracker in a mobile platform, being able to run correctly at real time (see Section 4.4 on page 42
- *Predict the movement from the target position.* Given a target position I elaborated a movement prediction based on that. This defined the behavior of the UAV and how it follows the target (see 5 on page 45).

Definitely, I got the tracker working on a mobile device correctly at real time despite the low capabilities of mobile platforms which allowed to test the tracker in a real device. The final application is now being tested in a real scenario remaining to do some adjustments on some parameters of the movement prediction.

During the development of the project I was able to interact with the state of art of deep learning, being able to experiment with it in a practical way taking it to real applications. I could also deal with hi-tech quadcopters for testing the results, having the chance to combine deep learning with robotics.

On the other hand, there is no any tracking model developed on a mobile platform source published yet so publishing the model I can help other developers to build its own applications, not only for UAV controlling but for any task they want. Apart from

that, the application itself can be also used for building bigger projects which involve object tracking.

Personally, I enjoyed very much carrying out this project as I could develop on my own a tracking system with the most advanced deep learning techniques. I was already involved in deep learning but I also found helpful this project for going deeper on it and know more about computer vision and specifically visual object trackers.

6.1 Future work

Given the good results got from the project, there are a few more steps that can be done from this point for improving it and achieve a final product ready for use by users:

- *Integrating the movement prediction in the model.* This is not really needed for getting a better product but it was one of the first objectives of the project and it could not been implemented finally. The intention was to generate a dataset labeling each target location (which would be the output of the current model) with a movement prediction, which corresponds to the formula explained in Section 2.1 on page 11. After that I would add a regression layer with 2 cells, each one responsible of predicting pitch and roll, respectively, and train this layer with the generated dataset.

- *Increasing the camera action degree.* Now the movement prediction works only when the quadcopter is flying high and it is following a target that is below it. This can be generalized to any position if we know the angle that the gimbal of the aircraft has and the application of some trigonometrical formulas. We can also consider that the target is further or nearer if the bounding box that encloses it gets smaller or bigger, respectively.

- *Build an specific tracking model for mobile platforms.* For achieving better results at tracking time in terms of frame rate and indirectly in accuracy, too (the more frames are tracked, the smother is the video sequence and the more accuracy are the results). As nowadays deep learning is getting closer to mobile platforms, there are new models build specifically for them that requires many less parameters than traditional models and run much faster than them. This models, like MoblieNet for example, performs special convolutions (like 1×1 convolutions)

and other stuff for making the networks faster. Building one of the trackers I used using a MobileNet as backend could improve hugely the performance of the tracker on the mobile devices, allowing the tracker work in older devices.

These measures, however, have not been included on the project due to the reduced time it has and the much effort they involve.

Appendices

A GOTURN code

A.1 Testing

```
import tensorflow as tf
import numpy as np
import sys
import time
import glob
from skimage.transform import resize
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from goturn import goturn_net

REGION_PADDING = 0.5
INPUT_SIZE = (227, 227)

def iou(pred, gt):
    x1s = np.fmax(pred[0], gt[0])
    x2s = np.fmin(pred[2], gt[2])
    y1s = np.fmax(pred[1], gt[1])
    y2s = np.fmin(pred[3], gt[3])
    ws = np.fmax(x2s - x1s, 0)
    hs = np.fmax(y2s - y1s, 0)
    intersection = ws * hs
    predArea = (pred[2] - pred[0]) * (pred[3] - pred[1])
    gtArea = (gt[2] - gt[0]) * (gt[3] - gt[1])
    union = np.fmax(predArea + gtArea - intersection, .00001)
    return intersection * 1.0 / union

def show_image(image):
    imgplot = plt.imshow(image)
    plt.show()

def relative_to_real(bbox, imsize):
    bbox = [x / 10 for x in bbox]
    bbox[0] = int(bbox[0] * imsize[0])
    bbox[1] = int(bbox[1] * imsize[1])
    bbox[2] = int(bbox[2] * imsize[0])
    bbox[3] = int(bbox[3] * imsize[1])
    return bbox

def get_crops(frames, bbox):
    assert(len(frames) == 2)

    bbox_w = bbox[2] - bbox[0]
    bbox_h = bbox[3] - bbox[1]

    region = [None] * 4
    region[0] = int(bbox[0] - REGION_PADDING * bbox_w)
    region[1] = int(bbox[1] - REGION_PADDING * bbox_h)
    region[2] = int(bbox[2] + REGION_PADDING * bbox_w)
    region[3] = int(bbox[3] + REGION_PADDING * bbox_h)

    h, w = frames[0].shape[:2]
    pad = max([-region[0], -region[1], region[2]-w, region[3]-h])
    if pad > 0:
        frames = [np.pad(f, [[pad, pad], [pad, pad], [0, 0]], 'mean') for f
                                                    in frames]
    else:
        pad = 0
```

```
    region = [x+pad for x in region]

    frames = [f[region[1]:region[3], region[0]:region[2]] for f in frames]
    frames = [resize(f, INPUT_SIZE) for f in frames]

    def converter(r_bbox):
        c_bbox = [b / 10 for b in r_bbox]
        c_bbox[0] = c_bbox[0] * (region[2] - region[0]) + region[0] - pad
        c_bbox[1] = c_bbox[1] * (region[3] - region[1]) + region[1] - pad
        c_bbox[2] = c_bbox[2] * (region[2] - region[0]) + region[0] - pad
        c_bbox[3] = c_bbox[3] * (region[3] - region[1]) + region[1] - pad

        return c_bbox


    return frames, converter


if __name__ == '__main__':
    t = goturn_net.TRACKNET(1, train=False)
    t.build()

    target = t.target
    image = t.image

    output = t.fc4

    with tf.Session() as sess:
        ckpt_dir = "./checkpoints"
        ckpt = tf.train.get_checkpoint_state(ckpt_dir)
        saver = tf.train.Saver()
        saver.restore(sess, ckpt.model_checkpoint_path)

        fnames = sorted(glob.glob('../data/*.jpg'))
        labels = open('../data/labels.txt').readlines()
        for i in range(len(labels)):
            labels[i] = np.asfarray(labels[i].split(' ')[1:], float)
        preds = []
        ious = []
        times = []
        continuous = True if "--continuous" in sys.argv else False

        assert(len(fnames) == len(labels))

        last_frame = mpimg.imread(fnames[0])
        last_bbox = labels[0]
        for f, label in zip(fnames[1:], labels[1:]):
            curr_frame = mpimg.imread(f)


            crops, converter = get_crops([last_frame, curr_frame], last_bbox)

            start = time.time()

            out = sess.run(output, feed_dict={
                target: crops[0][np.newaxis, ...] * 255,
                image: crops[1][np.newaxis, ...] * 255
            })

            end = time.time()

            out = np.array(converter(out[0]))

            preds += [out]
            ious += [iou(out, label)]
            times += [end-start]

            if continuous:
                last_bbox = out
            else:
```

```
                last_bbox = label

            last_frame = curr_frame

        print("Average IOU: {:.2f}".format(np.mean(ious)))
        print("Average time: {:.2f}".format(np.mean(times)))
        print("Average FPS: {:.2f}".format(1.0/np.mean(times)))
```

A.2 tfcoreml conversion

B Re^3 code

B.1 Testing

```
import tensorflow as tf
import numpy as np
import sys
import os
import time
import glob
from skimage.transform import resize
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

from re3.tracker import network
from re3.re3_utils.tensorflow_util import tf_util

REGION_PADDING = 0.5
MAX_STEPS = 32
INPUT_SIZE = (227, 227)

def iou(pred, gt):
    x1s = np.fmax(pred[0], gt[0])
    x2s = np.fmin(pred[2], gt[2])
    y1s = np.fmax(pred[1], gt[1])
    y2s = np.fmin(pred[3], gt[3])
    ws = np.fmax(x2s - x1s, 0)
    hs = np.fmax(y2s - y1s, 0)
    intersection = ws * hs
    predArea = (pred[2] - pred[0]) * (pred[3] - pred[1])
    gtArea = (gt[2] - gt[0]) * (gt[3] - gt[1])
    union = np.fmax(predArea + gtArea - intersection, .00001)
    return intersection * 1.0 / union

def relative_to_real(bbox, imsize):
    bbox = [x / 10 for x in bbox]
    bbox[0] = int(bbox[0] * imsize[0])
    bbox[1] = int(bbox[1] * imsize[1])
    bbox[2] = int(bbox[2] * imsize[0])
    bbox[3] = int(bbox[3] * imsize[1])
    return bbox

def get_crops(frames, bbox):
    assert(len(frames) == 2)

    bbox_w = bbox[2] - bbox[0]
    bbox_h = bbox[3] - bbox[1]

    region = [None] * 4
    region[0] = int(bbox[0] - REGION_PADDING * bbox_w)
    region[1] = int(bbox[1] - REGION_PADDING * bbox_h)
    region[2] = int(bbox[2] + REGION_PADDING * bbox_w)
    region[3] = int(bbox[3] + REGION_PADDING * bbox_h)

    h, w = frames[0].shape[:2]
    pad = max([-region[0], -region[1], region[2]-w, region[3]-h])
    if pad > 0:
        frames = [np.pad(f, [[pad, pad], [pad, pad], [0, 0]], 'mean') for f
                                                    in frames]
    else:
        pad = 0
    region = [x+pad for x in region]

    frames = [f[region[1]:region[3], region[0]:region[2]] for f in frames]
    frames = [resize(f, INPUT_SIZE) for f in frames]
```

```
    def converter(r_bbox):
        c_bbox = [b / 10 for b in r_bbox]
        c_bbox[0] = c_bbox[0] * (region[2] - region[0]) + region[0] - pad
        c_bbox[1] = c_bbox[1] * (region[3] - region[1]) + region[1] - pad
        c_bbox[2] = c_bbox[2] * (region[2] - region[0]) + region[0] - pad
        c_bbox[3] = c_bbox[3] * (region[3] - region[1]) + region[1] - pad

        return c_bbox


    return frames, converter

if __name__ == "__main__":
    image = tf.placeholder(tf.float32, [2, INPUT_SIZE[0], INPUT_SIZE[1], 3])
    prev_state = tuple([tf.placeholder(tf.float32, [1, 1024]) for _ in range(
                                            4)])


    output, state1, state2 = network.inference(image, 1, False, prevLstmState
                                            =prev_state)

    with tf.Session() as sess:
        ckpt = tf.train.get_checkpoint_state(os.path.join('re3', 'logs', '
                                                checkpoints'))
        tf_util.restore(sess, ckpt.model_checkpoint_path)

        fnames = sorted(glob.glob('../data/*.jpg'))
        labels = open('../data/labels.txt').readlines()
        for i in range(len(labels)):
            labels[i] = np.asfarray(labels[i].split(' ')[1:], float)
        preds = []
        ious = []
        times = []

        assert(len(fnames) == len(labels))

        last_frame = mpimg.imread(fnames[0])
        last_bbox = labels[0]
        last_state = [np.zeros([1, 1024]) for _ in range(4)]
        step = 0
        continuous = True if '--continuous' in sys.argv else False
        original_state = None

        for f, label in zip(fnames[1:], labels[1:]):
            curr_frame = mpimg.imread(f)

            crops, converter = get_crops([last_frame, curr_frame], last_bbox)

            start = time.time()

            if continuous:
                step += 1
                if step >= MAX_STEPS:
                    step = 0
                    last_state = original_state
                    crops[0] = crops[1]

            out, s1, s2 = sess.run([output, state1, state2], feed_dict={
                image: np.array([crops[0], crops[1]]) * 255,
                prev_state: last_state
            })
            last_state = [s1[0], s1[1], s2[0], s2[1]]

            if original_state is None:
                original_state = last_state

            end = time.time()

            out = np.array(converter(out[0]))
```

```
            preds += [out]
            ious += [iou(out, label)]
            times += [end-start]

            if continuous:
                last_bbox = out
            else:
                last_bbox = label
            last_frame = curr_frame

        print("Average IOU: {:.2f}".format(np.mean(ious)))
        print("Average time: {:.2f}".format(np.mean(times)))
        print("Average FPS: {:.2f}".format(1.0/np.mean(times)))
```

B.2 Keras

B.2.1 Keras custom layers

```
from keras.engine.topology import Layer
import keras.backend as K


class LRN2D(Layer):
    def __init__(self, alpha=2e-5, k=1, beta=0.75, n=5, **kwargs):
        if n % 2 == 0:
            raise NotImplementedError("LRN2D only works with odd n. n
                                                    provided: " + str(n))
        super(LRN2D, self).__init__(**kwargs)
        self.alpha = alpha
        self.k = k
        self.beta = beta
        self.n = n

    def build(self, input_shape):
        self.shape = input_shape
        super(LRN2D, self).build(input_shape)

    def call(self, x, mask=None):
        _, r, c, ch = x.shape
        b = 2
        half_n = self.n // 2
        input_sqr = K.square(x)
        extra_channels = K.zeros((b, r, c, ch + 2 * half_n))
        input_sqr = K.concatenate([extra_channels[:, :, :, :half_n],
                                   input_sqr,
                                   extra_channels[:, :, :, half_n + int(ch):]
                                                                        ],
                                  axis=3)

        scale = self.k
        norm_alpha = self.alpha / self.n
        for i in range(self.n):
            scale += self.alpha * input_sqr[:, :, :, i:i + int(ch)]
        scale = scale ** self.beta
        return x / scale

class CaffeLSTMCell(Layer):
    def __init__(self, units, **kwargs):
        self.units = units
        self.state_size = [self.units, self.units]
        super(CaffeLSTMCell, self).__init__(**kwargs)

    def build(self, input_shape):
        inputs_shape = input_shape[1] + self.units
        peephole_shape = inputs_shape + self.units
```

```
        self.block_input_W = self.add_weight(name='block_input_W',
                                             shape=[inputs_shape,
                                                    self.units],
                                             initializer='zeros')
        self.block_input_b = self.add_weight(name='block_input_b',
                                             shape=[self.units],
                                             initializer='zeros')

        self.input_gate_W = self.add_weight(name='input_gate_W',
                                            shape=[peephole_shape,
                                                   self.units],
                                            initializer='zeros')
        self.input_gate_b = self.add_weight(name='input_gate_b',
                                            shape=[self.units],
                                            initializer='zeros')

        self.forget_gate_W = self.add_weight(name='forget_gate_W',
                                             shape=[peephole_shape,
                                                    self.units],
                                             initializer='zeros')
        self.forget_gate_b = self.add_weight(name='forget_gate_b',
                                             shape=[self.units],
                                             initializer='zeros')

        self.output_gate_W = self.add_weight(name='output_gate_W',
                                             shape=[peephole_shape,
                                                    self.units],
                                             initializer='zeros')
        self.output_gate_b = self.add_weight(name='output_gate_b',
                                             shape=[self.units],
                                             initializer='zeros')

        self.built = True

    def call(self, inputs, states):
        cell_state_prev, cell_outputs_prev = states
        lstm_concat = K.concatenate([inputs,
                                     cell_outputs_prev], axis=1)
        peephole_concat = K.concatenate([lstm_concat, cell_state_prev], axis=
                                        1)

        block_input = K.tanh(K.dot(lstm_concat, self.block_input_W)
                             + self.block_input_b)

        input_gate = K.sigmoid(K.dot(peephole_concat, self.input_gate_W)
                               + self.input_gate_b)
        input_mult = input_gate * block_input

        forget_gate = K.sigmoid(K.dot(peephole_concat, self.forget_gate_W)
                                + self.forget_gate_b)
        forget_mult = forget_gate * cell_state_prev

        cell_state_new = input_mult + forget_mult
        cell_state_act = K.tanh(cell_state_new)

        output_concat = K.concatenate([lstm_concat, cell_state_new], axis=1)
        output_concat_shape = output_concat.get_shape().as_list()[1]

        output_gate = K.sigmoid(K.dot(output_concat, self.output_gate_W)
                                + self.output_gate_b)
        cell_outputs_new = output_gate * cell_state_act

        new_state = (cell_state_new, cell_outputs_new)

        return cell_outputs_new, new_state
```

B.2.2 Code

```
def build_net(return_lstm=True):
    input_shape = (1, 227*227*2, 3)

    input_layer = Input(batch_shape=input_shape)

    #div_batch = BatchReshape((2, 227, 227, 3))(input_layer)
    div_batch = Lambda(lambda x : K.reshape(x, (2, 227, 227, 3)),
                       name='div_batch')(input_layer)

    conv1 = Conv2D(name='conv1', filters=96, kernel_size=(11, 11),
                   strides=(4, 4), padding='valid',
                   activation='relu')(div_batch)
    pool1 = MaxPooling2D(name='pool1', pool_size=(3, 3), strides=(2, 2),
                         padding='valid')(conv1)
    lrn1 = LRN2D(name='norm1')(pool1)

    lrn1_1 = Lambda(lambda x : x[:, :, :, :48], name='split1l')(lrn1)
    lrn1_2 = Lambda(lambda x : x[:, :, :, 48:], name='split1r')(lrn1)

    conv2_g1 = Conv2D(name='conv2_g1', filters=128, kernel_size=(5, 5),
                      strides=(1, 1), padding='same', activation='relu')(
                                                                  lrn1_1)
    conv2_g2 = Conv2D(name='conv2_g2', filters=128, kernel_size=(5, 5),
                      strides=(1, 1), padding='same', activation='relu')(
                                                                  lrn1_2)
    conv2 = Concatenate(name='conv2', axis=3)([conv2_g1, conv2_g2])
    pool2 = MaxPooling2D(name='pool2', pool_size=(3, 3), strides=(2, 2),
                         padding='valid')(conv2)
    lrn2 = LRN2D(name='norm2')(pool2)

    conv3 = Conv2D(name='conv3', filters=384, kernel_size=(3, 3),
                   padding='same', activation='relu')(lrn2)

    conv3_1 = Lambda(lambda x : x[:, :, :, :192], name='split2l')(conv3)
    conv3_2 = Lambda(lambda x : x[:, :, :, 192:], name='split2r')(conv3)

    conv4_g1 = Conv2D(name='conv4_g1', filters=192, kernel_size=(3, 3),
                      strides=(1, 1), padding='same',
                      activation='relu')(conv3_1)
    conv4_g2 = Conv2D(name='conv4_g2', filters=192, kernel_size=(3, 3),
                      strides=(1, 1), padding='same',
                      activation='relu')(conv3_2)
    conv4 = Concatenate(name='conv4', axis=3)([conv4_g1, conv4_g2])

    conv4_1 = Lambda(lambda x : x[:, :, :, :192], name='split3l')(conv4)
    conv4_2 = Lambda(lambda x : x[:, :, :, 192:], name='split3r')(conv4)

    conv5_g1 = Conv2D(name='conv5_g1', filters=128, kernel_size=(3, 3),
                      strides=(1, 1), padding='same',
                      activation='relu')(conv4_1)
    conv5_g2 = Conv2D(name='conv5_g2', filters=128, kernel_size=(3, 3),
                      strides=(1, 1), padding='same',
                      activation='relu')(conv4_2)
    conv5 = Concatenate(name='conv5', axis=3)([conv5_g1, conv5_g2])
    pool5 = MaxPooling2D(name='pool5', pool_size=(3, 3), strides=(2, 2),
                                          padding='valid')(conv5)
    pool5 = Permute((3, 1, 2))(pool5)
    pool5_flat = Flatten()(pool5)


    conv1_skip = Conv2D(name='conv1_skip', filters=16, kernel_size=(1, 1))(
                                          lrn1)
    conv1_skip_prelu = PReLU(name='conv1_skip_prelu',
                             shared_axes=[1, 2])(conv1_skip)
    conv1_skip_perm = Permute((3, 1, 2))(conv1_skip_prelu)
    conv1_skip_flat = Flatten()(conv1_skip_perm) # Correct?
```

```
    conv2_skip = Conv2D(name='conv2_skip', filters=32, kernel_size=(1, 1))(
                                        lrn2)
    conv2_skip = PReLU(name='conv2_skip_prelu', shared_axes=[1, 2])(
                                        conv2_skip)
    conv2_skip = Permute((3, 1, 2))(conv2_skip)
    conv2_skip_flat = Flatten()(conv2_skip)

    conv5_skip = Conv2D(name='conv5_skip', filters=64,
                        kernel_size=(1, 1))(conv5)
    conv5_skip = PReLU(name='conv5_skip_prelu', shared_axes=[1, 2])(
                                        conv5_skip)
    conv5_skip = Permute((3, 1, 2))(conv5_skip)
    conv5_skip_flat = Flatten()(conv5_skip)

    big_concat = Concatenate(name='big_concat',
                             axis=1)([conv1_skip_flat,
                                      conv2_skip_flat,
                                      conv5_skip_flat,
                                      pool5_flat])

    big_concat = Lambda(lambda x: K.reshape(x, (1, 1, 74208)),
                        name='flatten_batch')(big_concat)

    fc6_out = Dense(2048, name='fc6', activation='relu')(big_concat)

    lstm1_cell = CaffeLSTMCell(1024)
    lstm2_cell = CaffeLSTMCell(1024)
    lstm1 = RNN(lstm1_cell, name='lstm1', stateful=True,
                return_state=return_lstm, return_sequences=True)
    lstm2 = RNN(lstm2_cell, name='lstm2', stateful=True,
                return_state=return_lstm, return_sequences=True)

    lstm1_out, s11, s12 = lstm1(fc6_out)


    lstm1_fc6 = Concat(name='lstm1_fc6', axis=-1)([fc6_out, lstm1_out])

    lstm2_out, s21, s22 = lstm2(lstm1_fc6)

    lstm2_out = Flatten()(lstm2_out)

    fc_output = Dense(4, name='fc_out')(lstm2_out)


    model = Model(input_layer, (fc_output, s11, s12, s21, s22))
    return model, lstm1, lstm2
```

B.2.3 Import weights

```
def load_model_weights(model, directory='tf-weights/'):
    conv1_W = np.load(directory+'re3_conv1_W_conv.npy')
    conv1_b = np.load(directory+'re3_conv1_b_conv.npy')
    conv1_skip_W = np.load(directory+'re3_conv1_skip_W_conv.npy')
    conv1_skip_b = np.load(directory+'re3_conv1_skip_b_conv.npy')
    conv1_skip_prelu = np.load(directory+'re3_conv1_skip_prelu.npy')
    conv2_W = np.load(directory+'re3_conv2_W_conv.npy')
    conv2_b = np.load(directory+'re3_conv2_b_conv.npy')
    conv2_skip_W = np.load(directory+'re3_conv2_skip_W_conv.npy')
    conv2_skip_b = np.load(directory+'re3_conv2_skip_b_conv.npy')
    conv2_skip_prelu = np.load(directory+'re3_conv2_skip_prelu.npy')
    conv3_W = np.load(directory+'re3_conv3_W_conv.npy')
    conv3_b = np.load(directory+'re3_conv3_b_conv.npy')
    conv4_W = np.load(directory+'re3_conv4_W_conv.npy')
    conv4_b = np.load(directory+'re3_conv4_b_conv.npy')
    conv5_W = np.load(directory+'re3_conv5_W_conv.npy')
    conv5_b = np.load(directory+'re3_conv5_b_conv.npy')
    conv5_skip_W = np.load(directory+'re3_conv5_skip_W_conv.npy')
```

```
conv5_skip_b = np.load(directory+'re3_conv5_skip_b_conv.npy')
conv5_skip_prelu = np.load(directory+'re3_conv5_skip_prelu.npy')
fc6_W = np.load(directory+'re3_fc6_W_fc.npy')
fc6_b = np.load(directory+'re3_fc6_b_fc.npy')
fc_out_W = np.load(directory+'re3_fc_output_W_fc.npy')
fc_out_b = np.load(directory+'re3_fc_output_b_fc.npy')
print("fc_out")
print(fc_out_W.shape, fc_out_b.shape)

lstm_1_bi_b = np.load(directory+'re3_lstm1_rnn_LSTM_block_input_biases.
                                        npy')
lstm_1_bi_W = np.load(directory+'re3_lstm1_rnn_LSTM_block_input_weights.
                                        npy')
lstm_1_fg_b = np.load(directory+'re3_lstm1_rnn_LSTM_forget_gate_biases.
                                        npy')
lstm_1_fg_W = np.load(directory+'re3_lstm1_rnn_LSTM_forget_gate_weights.
                                        npy')
lstm_1_ig_b = np.load(directory+'re3_lstm1_rnn_LSTM_input_gate_biases.npy
                                        ')
lstm_1_ig_W = np.load(directory+'re3_lstm1_rnn_LSTM_input_gate_weights.
                                        npy')
lstm_1_og_b = np.load(directory+'re3_lstm1_rnn_LSTM_output_gate_biases.
                                        npy')
lstm_1_og_W = np.load(directory+'re3_lstm1_rnn_LSTM_output_gate_weights.
                                        npy')

lstm_2_bi_b = np.load(directory+'re3_lstm2_rnn_LSTM_block_input_biases.
                                        npy')
lstm_2_bi_W = np.load(directory+'re3_lstm2_rnn_LSTM_block_input_weights.
                                        npy')
lstm_2_fg_b = np.load(directory+'re3_lstm2_rnn_LSTM_forget_gate_biases.
                                        npy')
lstm_2_fg_W = np.load(directory+'re3_lstm2_rnn_LSTM_forget_gate_weights.
                                        npy')
lstm_2_ig_b = np.load(directory+'re3_lstm2_rnn_LSTM_input_gate_biases.npy
                                        ')
lstm_2_ig_W = np.load(directory+'re3_lstm2_rnn_LSTM_input_gate_weights.
                                        npy')
lstm_2_og_b = np.load(directory+'re3_lstm2_rnn_LSTM_output_gate_biases.
                                        npy')
lstm_2_og_W = np.load(directory+'re3_lstm2_rnn_LSTM_output_gate_weights.
                                        npy')

model_names = [l.name for l in model.layers]

model.layers[model_names.index('conv1')].set_weights([conv1_W, conv1_b])

model.layers[model_names.index('conv1_skip')].set_weights([conv1_skip_W,
                                        conv1_skip_b])
model.layers[model_names.index('conv1_skip_prelu')].set_weights(
    [np.ones(model.layers[model_names.index('conv1_skip_prelu')].
                                            get_weights()[0].shape) *
                                            conv1_skip_prelu])

model.layers[model_names.index('conv2_g1')].set_weights([conv2_W[:, :, :,
                                        :128], conv2_b[:128]])
model.layers[model_names.index('conv2_g2')].set_weights([conv2_W[:, :, :,
                                        128:], conv2_b[128:]])

model.layers[model_names.index('conv2_skip')].set_weights([conv2_skip_W,
                                        conv2_skip_b])
model.layers[model_names.index('conv2_skip_prelu')].set_weights(
    [np.ones(model.layers[model_names.index('conv2_skip_prelu')].
                                            get_weights()[0].shape) *
                                            conv2_skip_prelu])

model.layers[model_names.index('conv3')].set_weights([conv3_W, conv3_b])

model.layers[model_names.index('conv4_g1')].set_weights([conv4_W[:, :, :,
                                        :192], conv4_b[:192]])
```

```
    model.layers[model_names.index('conv4_g2')].set_weights([conv4_W[:, :, :,
                                            192:], conv4_b[192:]])

    model.layers[model_names.index('conv5_g1')].set_weights([conv5_W[:, :, :,
                                            :128], conv5_b[:128]])
    model.layers[model_names.index('conv5_g2')].set_weights([conv5_W[:, :, :,
                                            128:], conv5_b[128:]])

    model.layers[model_names.index('conv5_skip')].set_weights([conv5_skip_W,
                                           conv5_skip_b])
    model.layers[model_names.index('conv5_skip_prelu')].set_weights(
        [np.ones(model.layers[model_names.index('conv5_skip_prelu')].
                                              get_weights()[0].shape) *
                                              conv5_skip_prelu])

    model.layers[model_names.index('fc6')].set_weights([fc6_W, fc6_b])
    model.layers[model_names.index('fc_out')].set_weights([fc_out_W, fc_out_b
                                            ])

    model.layers[model_names.index('lstm1')]
        .set_weights([lstm_1_bi_W, lstm_1_bi_b,
                      lstm_1_ig_W, lstm_1_ig_b,
                      lstm_1_fg_W, lstm_1_fg_b,
                      lstm_1_og_W, lstm_1_og_b])

    model.layers[model_names.index('lstm2')]
        .set_weights([lstm_2_bi_W, lstm_2_bi_b,
                      lstm_2_ig_W, lstm_2_ig_b,
                      lstm_2_fg_W, lstm_2_fg_b,
                      lstm_2_og_W, lstm_2_og_b])
    return model
```

B.2.4 Tracker test

```
import cv2
import time
import numpy as np

from utils import im_util, bb_util
from model import network

LSTM_SIZE = 1024
CROP_SIZE = 227
CROP_PAD = 2
MAX_TRACE_LENGTH = 32
SPEED_OUTPUT = True

IMAGENET_MEAN = np.array([123.151630838, 115.902882574, 103.062623801],
                         dtype='float32')

class Tracker():
    def __init__(self):
        self.past_bbox = None
        self.lstm_state = None
        self.prev_image = None
        self.forward_count = None
        self.total_forward_count = 0
        self.original_features = None
        self.time = 0
        model, self.lstm1, self.lstm2 = network.build_net(return_lstm=True)
        model.summary()
        self.model = network.load_model_weights(model)

    def track(self, image, starting_box=None):
        start_time = time.time()

        if type(image) == str:
```

```
        image = cv2.imread(image)[:,:,::-1]
    else:
        image = image.copy()

    original_image = image.copy()

    image_read_time = time.time() - start_time

    # if starting_box != None -> start new tracking
    if starting_box is not None:
        self.past_bbox = np.array(starting_box)
        self.lstm_state = [np.zeros((1, 1024)) for _ in range(4)]
        self.prev_image = image
        self.forward_count = 0
        self.original_features = None
    else:
        if self.past_bbox is None:
            print("Error, no starting box specified")
            return

    cropped_input0, past_bbox_padded = im_util.get_cropped_input(
        self.prev_image,
        self.past_bbox,
        CROP_PAD,
        CROP_SIZE
    )

    cropped_input1, _ = im_util.get_cropped_input(
        image,
        self.past_bbox,
        CROP_PAD,
        CROP_SIZE
    )

    self.lstm1.reset_states(states=self.lstm_state[:2])
    self.lstm2.reset_states(states=self.lstm_state[2:])

    raw_output, s11, s12, s21, s22 = self.model.predict(
        np.array(np.reshape([cropped_input0, cropped_input1],
                            (1, -1, 3)))-IMAGENET_MEAN
    )

    self.lstm_state = [s11, s12, s21, s22]

    print(raw_output)
    print(self.lstm_state)

    if self.forward_count == 0:
        self.original_features = [s11, s12, s21, s22]

    self.prev_image = image.copy()

    output_bbox = bb_util.from_crop_coordinate_system(
        raw_output.squeeze() / 10.0,
        past_bbox_padded, 1, 1
    )

    if self.forward_count > 0 and self.forward_count%MAX_TRACE_LENGTH ==
                                                0:
        cropped_input, _ = im_util.get_cropped_input(image,
                                                     output_bbox,
                                                     CROP_PAD,
                                                     CROP_SIZE)

        inp = np.tile(cropped_input[np.newaxis, ...], (2,1,1,1))

        self.lstm1.reset_states(states=(self.original_features[0],
                                        self.original_features[1]))
        self.lstm2.reset_states(states=(self.original_features[2],
                                        self.original_features[3]))
```

```
            raw_output, s11, s12, s21, s22 = self.model.predict(
                np.array(np.reshape(inp, (1, -1, 3)))-IMAGENET_MEAN
            )
            s11, s12, s21, s22 = s11, s12, s21, s22

            self.lstm_state = [s11, s12, s21, s22]

        self.forward_count += 1
        self.total_forward_count += 1

        if starting_box is not None:
            output_bbox = np.array(starting_box)

        self.past_bbox = output_bbox

        end_time = time.time()

        if self.total_forward_count > 0:
            self.time += (end_time - start_time - image_read_time)

        if SPEED_OUTPUT and self.total_forward_count % 100 == 0:
            print('Current tracking speed:   %.3f FPS' %
                  (2 * 1 / (end_time - start_time - image_read_time)))
            print('Current image read speed: %.3f FPS' %
                  (1 / (image_read_time)))
            print('Mean tracking speed:      %.3f FPS\n' %
                  (2 * self.total_forward_count / max(.00001, self.time)))

        return output_bbox


if __name__ == '__main__':
    t = Tracker()
    data_dir = 'data/'
    bbox = [175.00, 154.00, 251.00, 229.00]

    files = sorted(glob.glob(data_dir + '/*.jpg'))
    t.track(files[0], starting_box=bbox)

    for f in files:
        image = cv2.imread(f)
        imageRGB = image[:,:,::-1]

        bbox = t.track(imageRGB)
        print(bbox)
        cv2.rectangle(image,
                (int(bbox[0]), int(bbox[1])),
                (int(bbox[2]), int(bbox[3])),
                [0, 0, 255], 2)
        cv2.imwrite('out/'+f.split('/')[1], image)
```

B.2.5 Converter

```
import coremltools
from coremltools.proto import NeuralNetwork_pb2

from model import network

if __name__ == '__main__':
    model = network.build_net()
    model = network.load_model_weights(model)

    def convert_lambda(layer):
        params = NeuralNetwork_pb2.CustomLayerParams()
        if layer.name == 'div_batch':
            params.className = "DivBatch"
```

```
            params.description = "Reshapes a tensor including the batch axis.
                                                  "
        elif layer.name == 'flatten_batch':
            params.className = "FlattenBatch"
            params.description = "Flattens a tensor including the batch axis.
                                                  "
        elif layer.name == 'concat_axis':
            params.className = "ConcatAxis"
            params.description = "Concatenates a list of tensors on last axis
                                                  ."
        else:
            params.className = "Split"
            params.description = "Splits the tensor by half in axis 3."

            if layer.name == 'split1l':
                params.parameters['from'].intValue = 0
                params.parameters['to'].intValue = 48
            elif layer.name == 'split1r':
                params.parameters['from'].intValue = 48
                params.parameters['to'].intValue = 96
            elif layer.name in ['split2l', 'split3l']:
                params.parameters['from'].intValue = 0
                params.parameters['to'].intValue = 192
            elif layer.name in ['split2r', 'split3r']:
                params.parameters['from'].intValue = 192
                params.parameters['to'].intValue = 384

        return params

    def convert_LRN2D(layer):
        params = NeuralNetwork_pb2.CustomLayerParams()
        params.className = "LRN2D"
        params.description = "Local Response Normalization Layer"

        return params

    def convert_CaffeLSTM(layer):
        params = NeuralNetwork_pb2.CustomLayerParams()
        params.parameters['units'].intValue = 1024;
        params.className = "CaffeLSTM"
        params.description = "Caffe LSTM cell implementation."

        for weights in layer.get_weights():
            w = params.weights.add()
            w.floatValue.extend(map(float, weights.flatten().tolist()))
            print(weights.shape)

        return params


    coreml_base = coremltools.converters.keras.convert(
        model,
        input_names=["flattened_image",
                     "lstm1_state1",
                     "lstm1_state2",
                     "lstm2_state1",
                     "lstm2_state2"],
        output_names=["output",
                      "lstm1_new_state1",
                      "lstm1_new_state2",
                      "lstm2_new_state1",
                      "lstm2_new_state2"],
        add_custom_layers=True,
        custom_conversion_functions={"LRN2D": convert_LRN2D
                                     "RNN": convert_CaffeLSTM}

    coreml_base.author = "emascandela"
    coreml_base.license = "Public Domain"
    coreml_base.short_description = "RE3 tracker model"
```

```
    coreml_base.save("Model.mlmodel")
```

B.2.6 CoreML custom layers

B.2.6.1 Split

```
import Foundation
import CoreML

@objc(Split) class Split: NSObject, MLCustomLayer {
    var m = MultiArray<Float>(shape: [3, 4, 2])

    let from: Int
    let to: Int

    required init(parameters: [String : Any]) throws {
       if let from = parameters["from"] as? Int {
            self.from = from
        } else {
            self.from = 0
        }
        if let to = parameters["to"] as? Int {
            self.to = to
        } else {
            self.to = 0
        }

        print("Split:", #function, parameters)

        super.init()
    }

    func setWeightData(_ weights: [Data]) throws {

    }

    func outputShapes(forInputShapes inputShapes: [[NSNumber]]) throws -> [[
                                            NSNumber]] {
        // print(#function, inputShapes)
        var shape = inputShapes
        let c = shape[0][shape[0].count-3].intValue / 2
        shape[0][shape[0].count-3] = NSNumber(value: c)
        print("Split:", #function, inputShapes, shape)
        return shape
    }

    func evaluate(inputs: [MLMultiArray], outputs: [MLMultiArray]) throws {
        print(#function, inputs.count, outputs.count)
        print(#function, inputs[0].shape, outputs[0].shape)

        for i in 0..<inputs.count {
            var input = MultiArray<Float>(inputs[i])
            var output = MultiArray<Float>(outputs[i])


            let shape = input.shape
            for b in 0..<shape[1] {
                for c in self.from..<self.to {
                    for h in 0..<shape[3] {
                        for w in 0..<shape[4] {
                            output[0, b, c-self.from, h, w] = input[0, b, c,
                                                                    h, w]
                        }
                    }
                }
            }
```

```
            print(self.from, self.to)
            let last = output.shape[4]-1
            print(output[0, 0, 0, 0, 0])
            print(output[0, 1, 0, 0, 0])
            print(output[0, 0, 0, last, last])
            print(output[0, 1, 0, last, last])
            print(output[0, 0, 5, 12, 2])
            print(output[0, 0, 7, 4, 8])
            print(output[0, 1, 5, 12, 2])
            print(output[0, 1, 7, 4, 8])

        }
    }
}
```

B.2.6.2 Concat

```
import Foundation
import CoreML

@objc(CustomConcat) class CustomConcat: NSObject, MLCustomLayer {
    required init(parameters: [String : Any]) throws {
        print(#function, parameters)

        super.init()
    }

    func setWeightData(_ weights: [Data]) throws {

    }

    func outputShapes(forInputShapes inputShapes: [[NSNumber]]) throws -> [[
                                                    NSNumber]] {
        // input shape = (1, 227*227*2, 3)
        // output shape = (2, 3, 227, 227)

        let s = NSNumber(value: 1)
        let b = NSNumber(value: 1)
        let ch = NSNumber(value: inputShapes[0][2].intValue + inputShapes[1][
                                                    2].intValue)
        let h = NSNumber(value: 1)
        let w = NSNumber(value: 1)

        let shape: [NSNumber] = [s, b, ch, h, w]
        print("Concat:",  #function, inputShapes, shape)

        return [shape, shape]
    }

    func evaluate(inputs: [MLMultiArray], outputs: [MLMultiArray]) throws {
        print(#function, inputs.count, outputs.count)
        print(#function, inputs[0].shape, outputs[0].shape)

        var input1 = MultiArray<Float>(inputs[0])
        var input2 = MultiArray<Float>(inputs[0])

        assert(input1.shape[0] == input2.shape[0])
        assert(input1.shape[1] == input2.shape[1])
        assert(input1.shape[3] == input2.shape[3])
        assert(input1.shape[4] == input2.shape[4])

        for i in 0..<outputs.count {
            var output = MultiArray<Float>(outputs[i])

            for s in 0..<input1.shape[0] {
                for b in 0..<input1.shape[1] {
                    for c in 0..<input1.shape[2] {
```

```
                        for h in 0..<input1.shape[3] {
                            for w in 0..<input1.shape[4] {
                                output[s, b, c, h, w] = input1[s, b, c, h, w]
                            }
                        }
                    }
                }
            }
            for s in 0..<input2.shape[0] {
                for b in 0..<input2.shape[1] {
                    for c in 0..<input2.shape[2] {
                        for h in 0..<input2.shape[3] {
                            for w in 0..<input2.shape[4] {
                                output[s, b, c+input1.shape[2], h, w] =
                                                                    input2
                                                                    [s, b,
                                                                     c, h,
                                                                     w]
                            }
                        }
                    }
                }
            }
            print(output[0, 0, 0, 0, 0])
            print(output[0, 0, output.shape[2]-1, output.shape[3]-1, output.
                                                    shape[4]-1])
        }
    }
}
```

B.2.6.3 Flatten

```
import Foundation
import CoreML

@objc(FlattenBatch) class FlattenBatch: NSObject, MLCustomLayer {
    required init(parameters: [String : Any]) throws {
        print(#function, parameters)

        super.init()
    }

    func setWeightData(_ weights: [Data]) throws {

    }

    func outputShapes(forInputShapes inputShapes: [[NSNumber]]) throws -> [[
                                                    NSNumber]] {
        // input shape = (1, 2, 37104, 1, 1)
        // output shape = (1, 1, 74208, 1, 1)

        let s = NSNumber(value: 1)
        let b = NSNumber(value: 1)
        let ch = NSNumber(value: 74208)
        let h = NSNumber(value: 1)
        let w = NSNumber(value: 1)

        let shape: [[NSNumber]] = [[s, b, ch, h, w]]

        print("FlattenBatch:", #function, inputShapes, shape)

        return shape
    }

    func evaluate(inputs: [MLMultiArray], outputs: [MLMultiArray]) throws {
        print(#function, inputs.count, outputs.count)
        print(#function, inputs[0].shape, outputs[0].shape)
```

```
for idx in 0..<inputs.count {
    var input = MultiArray<Float>(inputs[idx])//.reshaped([74208])
    var output = MultiArray<Float>(outputs[idx])


    var permute2 = MultiArray<Float>(shape: [1, 2, 27, 27, 16]).
                                        reshaped([1, 2, 27*16*27,
                                        1, 1])
    for i in 0..<11664 {
        permute2[0, 0, i, 0, 0] = input[0, 0, i, 0, 0]
        permute2[0, 1, i, 0, 0] = input[0, 1, i, 0, 0]
    }
    permute2 = permute2.reshaped([1, 2, 27, 27, 16]).transposed([0, 1
                                        , 4, 2, 3])


    var permute3 = MultiArray<Float>(shape: [1, 2, 13, 13, 32]).
                                        reshaped([1, 2, 13*32*13,
                                        1, 1])
    for i in 0..<5408 {
        permute3[0, 0, i, 0, 0] = input[0, 0, i+11664, 0, 0]
        permute3[0, 1, i, 0, 0] = input[0, 1, i+11664, 0, 0]
    }
    permute3 = permute3.reshaped([1, 2, 13, 13, 32]).transposed([0, 1
                                        , 4, 2, 3])


    var permute4 = MultiArray<Float>(shape: [1, 2, 13, 13, 64]).
                                        reshaped([1, 2, 13*64*13,
                                        1, 1])
    for i in 0..<10816 {
        permute4[0, 0, i, 0, 0] = input[0, 0, i+5408+11664, 0, 0]
        permute4[0, 1, i, 0, 0] = input[0, 1, i+5408+11664, 0, 0]
    }
    permute4 = permute4.reshaped([1, 2, 13, 13, 64]).transposed([0, 1
                                        , 4, 2, 3])


    var permute1 = MultiArray<Float>(shape: [1, 2, 6, 6, 256]).
                                        reshaped([1, 2, 6*256*6, 1
                                        , 1])
    for i in 0..<9216 {
        permute1[0, 0, i, 0, 0] = input[0, 0, i+5408+11664+10816, 0,
                                            0]
        permute1[0, 1, i, 0, 0] = input[0, 1, i+5408+11664+10816, 0,
                                            0]
    }
    permute1 = permute1.reshaped([1, 2, 6, 6, 256]).transposed([0, 1,
                                         4, 2, 3])
    print("PERMUTE1:")
    print(permute1[0, 0, 0, 0, 0])
    print(permute1[0, 0, 0, 0, 1])
    print(permute1[0, 0, 0, 1, 0])
    print(permute1[0, 0, 1, 0, 0])
    print(permute1[0, 1, 0, 0, 0])

    //permute2 = permute2.reshaped([1, 2, 11664, 1, 1])
    for b in 0..<permute2.shape[1] {
        for c in 0..<permute2.shape[2] {
            for h in 0..<permute2.shape[3] {
                for w in 0..<permute2.shape[4] {
                    let i = c * permute2.shape[4]*permute2.shape[3] +
                        h * permute2.shape[4] + w + b * 37104
                    output[0, 0, i, 0, 0] = permute2[0, b, c, h, w]
                }
            }
        }
    }
```

```
            for b in 0..<permute3.shape[1] {
                for c in 0..<permute3.shape[2] {
                    for h in 0..<permute3.shape[3] {
                        for w in 0..<permute3.shape[4] {
                            let i = c * permute3.shape[4]*permute3.shape[3] +
                                h * permute3.shape[4] + w + b * 37104 + 11664
                            output[0, 0, i, 0, 0] = permute3[0, b, c, h, w]
                        }
                    }
                }
            }

            for b in 0..<permute4.shape[1] {
                for c in 0..<permute4.shape[2] {
                    for h in 0..<permute4.shape[3] {
                        for w in 0..<permute4.shape[4] {
                            let i = c * permute4.shape[4]*permute4.shape[3] +
                                h * permute4.shape[4] + w + b * 37104 + 11664
                                                                          +
                                                                         5408
                            output[0, 0, i, 0, 0] = permute4[0, b, c, h, w]
                        }
                    }
                }
            }

            for b in 0..<permute1.shape[1] {
                for c in 0..<permute1.shape[2] {
                    for h in 0..<permute1.shape[3] {
                        for w in 0..<permute1.shape[4] {
                            let i = c * permute1.shape[4]*permute1.shape[3] +
                                h * permute1.shape[4] + w + b * 37104 + 11664
                                                                          +
                                                                         5408 +
                                                                          10816
                            output[0, 0, i, 0, 0] = permute1[0, b, c, h, w]
                        }
                    }
                }
            }

            print(output)

        }
    }
}
```

B.2.6.4 LRN

```
import Foundation
import CoreML
import Accelerate

@objc(LRN2D) class LRN2D: NSObject, MLCustomLayer {
    let alpha: Float
    let beta: Float
    let k: Float
    let n: Int

    required init(parameters: [String : Any]) throws {
        self.alpha = 2e-5
        self.beta = 0.75
        self.k = 1.0
        self.n = 5

        print(#function, parameters)
```

```
        super.init()
    }

    func setWeightData(_ weights: [Data]) throws {

    }

    func outputShapes(forInputShapes inputShapes: [[NSNumber]]) throws -> [[
                                                NSNumber]] {
        let shapes = inputShapes
        print("LRN:", #function, shapes)
        return shapes
    }

    func evaluate(inputs: [MLMultiArray], outputs: [MLMultiArray]) throws {
        print(#function, inputs.count, outputs.count)
        print(#function, inputs[0].shape, outputs[0].shape)
        for idx in 0..<inputs.count {
            let input = inputs[idx]
            let output = outputs[idx]

            let shape = input.shape
            let b = shape[1]
            let c = shape[2]
            let h = shape[3]
            let w = shape[4]

            var pointerA: UnsafeMutablePointer<Float>
            var pointerB: UnsafeMutablePointer<Float>

            let half_n: Int = n / 2

            let data = try MLMultiArray(shape: [b, c, h, w], dataType: .
                                                float32)

            let count: Int = input.count
            for i in 0..<count {
                data[i] = input[i]
            }

            let data_sqr = try MLMultiArray(shape: data.shape, dataType: .
                                                float32)

            pointerA = UnsafeMutablePointer<Float>(OpaquePointer(data.
                                                dataPointer))
            pointerB = UnsafeMutablePointer<Float>(OpaquePointer(data_sqr.
                                                dataPointer))
            vDSP_vsq(pointerA, 1, pointerB, 1, vDSP_Length(data_sqr.count))
            //vvsqr(pointerB, pointerA, &cnt)

            let extra_channels = try MLMultiArray(shape: [b, NSNumber(value:
                                                c.intValue + 2 * half_n),
                                                       h, w], dataType: .




            for i in 0..<data_sqr.shape[0].intValue {
                for j in 0..<data_sqr.shape[1].intValue {
                    for k in 0..<data_sqr.shape[2].intValue {
                        for l in 0..<data_sqr.shape[3].intValue {
                            let index1: [NSNumber] = [NSNumber(value: i),
                                                      NSNumber(value: j),
                                                      NSNumber(value: k),
                                                      NSNumber(value: l)]

                            let index2: [NSNumber] = [NSNumber(value: i),
                                                      NSNumber(value: j +
```

```
                                                NSNumber(value: k),
                                                NSNumber(value: l)]

                        extra_channels[index2] = data_sqr[index1]
                    }
                }
            }
        }

        //for i in 0..<extra_channels.count {
        //    print(extra_channels[i])
        //}

        let scale = try MLMultiArray(shape: data.shape, dataType: .
                                                    float32)
        for i in 0..<scale.count {
            scale[i] = NSNumber(value: k)
        }

        for i in 0..<n {
            for j in 0..<b.intValue {
                for k in 0..<c.intValue {
                    for l in 0..<h.intValue {
                        for m in 0..<w.intValue {
                            let index1: [NSNumber] = [NSNumber(value: j),
                                                      NSNumber(value: k),
                                                      NSNumber(value: l),
                                                      NSNumber(value: m)]

                            let index2: [NSNumber] = [NSNumber(value: j),
                                                      NSNumber(value: k+i



                                                      NSNumber(value: l),
                                                      NSNumber(value: m)]

                            scale[index1] =
                                NSNumber(
                                    value: scale[index1].floatValue+alpha
                                    *extra_channels[index2].floatValue)
                        }
                    }
                }
            }
        }

        for i in 0..<b.intValue {
            for j in 0..<c.intValue {
                for k in 0..<h.intValue {
                    for l in 0..<w.intValue {
                        let index1: [NSNumber] = [NSNumber(value: i),
                                                  NSNumber(value: j),
                                                  NSNumber(value: k),
                                                  NSNumber(value: l)]

                        let index2: [NSNumber] = [NSNumber(value: 0),
                                                  NSNumber(value: i),
                                                  NSNumber(value: j),
                                                  NSNumber(value: k),
                                                  NSNumber(value: l)]

                        var value = input[index2].floatValue
                        value /= pow(scale[index1].floatValue, beta)
                        output[index2] = NSNumber(value: value)
                    }
                }
            }
```

```
            }
            for i in 0..<10 {
                print(String(format:"%.025f", MultiArray<Float>(input)[0,0, 2
                                                        , 0, i]))
            }
        }
    }
}
```

B.2.6.5 Caffe LSTM

```
import Foundation
import CoreML
import Accelerate
import Metal

@objc(CaffeLSTM) class CaffeLSTM: NSObject, MLCustomLayer {
    var states: [MultiArray<Float>]

    var block_input_b: MultiArray<Float> = MultiArray<Float>(shape: [0])
    var block_input_W: MultiArray<Float> = MultiArray<Float>(shape: [0])
    var forget_gate_b: MultiArray<Float> = MultiArray<Float>(shape: [0])
    var forget_gate_W: MultiArray<Float> = MultiArray<Float>(shape: [0])
    var input_gate_b: MultiArray<Float> = MultiArray<Float>(shape: [0])
    var input_gate_W: MultiArray<Float> = MultiArray<Float>(shape: [0])
    var output_gate_b: MultiArray<Float> = MultiArray<Float>(shape: [0])
    var output_gate_W: MultiArray<Float> = MultiArray<Float>(shape: [0])

    let units: Int

    required init(parameters: [String : Any]) throws {
        if let units = parameters["units"] as? Int {
            self.units = units
        } else {
            self.units = 0
        }

        states = [MultiArray<Float>(shape: [1, 1024], initial: 0),
                  MultiArray<Float>(shape: [1, 1024], initial: 0)]

        super.init()
    }

    func readByte(_ idx: Int, _ data: Data) -> Float32 {
        var res: Float32 = 0.0
        let pt = UnsafeMutablePointer<UInt8>.allocate(capacity: 4)

        data.copyBytes(to: pt, from: idx*4..<(idx+1)*4)
        memccpy(&res, pt, 4, 4)

        return res
    }

    func loadWeight(_ data: Data) -> MultiArray<Float> {
        let shape = [data.count/(4*1024), 1024]

        var w = MultiArray<Float>(shape: shape)
        w = w.reshaped([shape.reduce(1, {$0 * $1})])
        //print("Loading...")
        for i in 0..<w.count {
            w[i] = readByte(i, data)
            // if i < 5 || i > w.count-5 {
            //     print(w[i])
            // }
        }

        w = w.reshaped(shape)
```

```
        return w
    }

    func setWeightData(_ weights: [Data]) throws {
        block_input_W = loadWeight(weights[0])
        block_input_b = loadWeight(weights[1])
        input_gate_W = loadWeight(weights[2])
        input_gate_b = loadWeight(weights[3])
        forget_gate_W = loadWeight(weights[4])
        forget_gate_b = loadWeight(weights[5])
        output_gate_W = loadWeight(weights[6])
        output_gate_b = loadWeight(weights[7])
    }

    func outputShapes(forInputShapes inputShapes: [[NSNumber]]) throws -> [[
                                                NSNumber]] {
        // input shape = (1, 227*227*2, 3)
        // output shape = (1, 1, 74208)

        let s = NSNumber(value: 1)
        let b = NSNumber(value: 1)
        let ch = NSNumber(value: 1024)
        let h = NSNumber(value: 1)
        let w = NSNumber(value: 1)

        let shape: [[NSNumber]] = [[s, b, ch, h, w]]

        print("LSTM:", #function, inputShapes, shape)

        return shape
    }

    func evaluate(inputs: [MLMultiArray], outputs: [MLMultiArray]) throws {
        print(#function, inputs.count, outputs.count)
        print(#function, inputs[0].shape, outputs[0].shape)
        print(block_input_W[0, 0])
        print(block_input_W[0, 1])
        print(block_input_W[1, 0])
        print(block_input_W[1, 1])


        for i in 0..<inputs.count {
            var input = MultiArray<Float>(inputs[i])

            input = input.reshaped([1, input.shape[2]])
            var output = MultiArray<Float>(outputs[i])
            let output_shape = output.shape

            let cell_state_prev = states[0]
            let cell_output_prev = states[1]
            print("cell_state", cell_state_prev.shape)
            print("cell_output", cell_output_prev.shape)

            let lstm_concat = concat(input, cell_output_prev)
            for i in 0..<10 {
                print(String(format:"%.025f", block_input_W[0, i]))
            }
            print(lstm_concat[0, 0], lstm_concat[0, 1])
            print(block_input_W[0, 0], block_input_W[0, 1])

            print("lstm_concat", lstm_concat.shape)
            let peephole_concat = concat(lstm_concat, cell_state_prev)
            print("peephole", peephole_concat.shape)

            let block_input = tanh(dot(lstm_concat, self.block_input_W) +
                                                    self.block_input_b)
            //
            // print("block_input_W", self.block_input_W.shape)
            print("block_input", block_input.shape)
            //print(block_input)
```

```
            let input_gate = sigmoid(dot(peephole_concat, self.input_gate_W)
                                                    + self.input_gate_b)
            print("input_gate", input_gate.shape)
            let input_mult = input_gate * block_input


            let forget_gate = sigmoid(dot(peephole_concat, self.forget_gate_W
                                                    ) + self.forget_gate_b)
            let forget_mult = forget_gate * cell_state_prev

            let cell_state_new = input_mult + forget_mult
            let cell_state_act = tanh(cell_state_new)

            let output_concat = concat(lstm_concat, cell_state_new)

            let output_gate = sigmoid(dot(output_concat, self.output_gate_W)
                                                    + self.output_gate_b)
            let cell_outputs_new = output_gate * cell_state_act

            states = [cell_state_new, cell_outputs_new]
            output = output.reshaped(cell_outputs_new.shape)

            for y in 0..<output.shape[0] {
                for x in 0..<output.shape[1] {
                    output[y, x] = cell_outputs_new[y, x]
                }
            }

            output = output.reshaped(output_shape)
            //print(output)
            let d = dot(lstm_concat, self.block_input_W)
            for i in 0..<10 {
                print(String(format:"%.025f", input[0, i]))
            }
        }
    }
}
```

B.2.6.6 Print

```
import CoreML

@objc(Print) class Print: NSObject, MLCustomLayer {
    required init(parameters: [String : Any]) throws {

    }

    func setWeightData(_ weights: [Data]) throws {

    }

    func outputShapes(forInputShapes inputShapes: [[NSNumber]]) throws -> [[
                                            NSNumber]] {
        return inputShapes
    }

    func evaluate(inputs: [MLMultiArray], outputs: [MLMultiArray]) throws {
        print("Print")

        for i in 0..<inputs.count {
            let input = inputs[i]
            let output = outputs[i]

            var inp = MultiArray<Float>(input)
            inp = inp.reshaped([inp.shape.reduce(1, {$0*$1})])
            var out = MultiArray<Float>(output)
```

```
            out = out.reshaped([out.shape.reduce(1, {$0*$1})])
            for i in 0..<inp.shape[0] {
                if i < 3 || i > inp.shape[0]-3 {
                    print(inp[i])
                }
                out[i] = inp[i]
            }
        }
    }
}
```

C ROLO code

C.1 ROLO testing

```
import tensorflow as tf
import numpy as np
import cv2
import time
from PIL import Image
import glob

import rolo.third_party.YOLO_network as yolo
import rolo.experiments.training.ROLO_step3_train_30_exp2 as rolo

def iou(pred_, gt_):
    pred = np.array(pred_) * 447
    gt = np.array(gt_) * 447

    pred[2] += pred[0]
    pred[3] += pred[1]
    gt[2] += gt[0]
    gt[3] += gt[1]

    x1s = np.fmax(pred[0], gt[0])
    x2s = np.fmin(pred[2], gt[2])
    y1s = np.fmax(pred[1], gt[1])
    y2s = np.fmin(pred[3], gt[3])
    ws = np.fmax(x2s - x1s, 0)
    hs = np.fmax(y2s - y1s, 0)
    intersection = ws * hs
    predArea = (pred[2] - pred[0]) * (pred[3] - pred[1])
    gtArea = (gt[2] - gt[0]) * (gt[3] - gt[1])
    union = np.fmax(predArea + gtArea - intersection, .00001)
    return intersection * 1.0 / union


if __name__ == '__main__':
    yolo_net = yolo.YOLO_TF()
    rolo_net = rolo.ROLO_TF()
    rolo_net.build_networks()


    fnames = sorted(glob.glob('../data/*.jpg'))
    gt = open('../data/labels.txt').readlines()
    fnames = fnames[:20]
    gt = gt[:20]

    with Image.open(fnames[0]) as img:
        img_size = np.asarray(img.size)
        print(img_size)

    for i in range(len(gt)):
        gt[i] = np.array(gt[i].split(' ')[1:], float)
        gt[i][2] = (gt[i][2]-gt[i][0])
        gt[i][3] = (gt[i][3]-gt[i][1])

    num_steps = 3

    last_location = gt[0]
    last_state = np.zeros((self.batch_size, 2*self.num_input))

    for fname, label in zip(fnames[1:], gt[1:]):
        img = yolo_net.file_to_img(fname)

        # Pass through YOLO layers
        h_img, w_img,_ = img.shape
```

```
        img_resized = cv2.resize(img, (448, 448))
        img_RGB = cv2.cvtColor(img_resized,cv2.COLOR_BGR2RGB)
        img_resized_np = np.asarray( img_RGB )
        inputs = np.zeros((1,448,448,3),dtype='float32')
        inputs[0] = (img_resized_np/255.0)*2.0-1.0
        in_dict = {yolo_net.x : inputs}

        start_time = time.time()
        feature= yolo_net.sess.run(yolo_net.fc_30, feed_dict=in_dict)
        output = yolo_net.sess.run(yolo_net.fc_32,feed_dict=in_dict)  # make
                                                sure it does not run conv
                                                layers twice

        locations = yolo_net.interpret_output(output[0])
        location = yolo_net.find_best_location(locations, label) # find the
                                                ROI that has the maximum IOU
                                                with the ground truth

        # change location into [0, 1]
        location = yolo_net.location_from_0_to_1(w_img, h_img, location)
        yolo_output =
            np.concatenate((np.reshape(feature, [-1, yolo_net.num_feat]),
                            np.reshape(location, [-1, yolo_net.num_predict])),
                            axis=1)

        label = yolo_net.location_from_0_to_1(w_img, h_img, label)

        yolo_output = np.reshape(yolo_oytput, [1, num_steps, num_input])
        label = np.reshape(label, [1, 4])

        pred_location, state = sess.run(rolo_net.pred_location
                                        rolo_net.state,
                                        feed_dict={
            self.x: batch_xs,
            self.y: batch_ys,
            self.istate: last_state
        })

        t = time.time() - start_time

        times.append[t]
        outputs.append[pred_location[0]]
        ious.append[iou(pred_locations[0], label)]

        last_state = state
        last_location = pred_location[0]


        if not continuous:
            last_location = label


    print("Average IOU: {:.2f}".format(np.mean(ious)))
    print("Average time: {:.2f}".format(np.mean(times)))
    print("Average FPS: {:.2f}".format(1.0/np.mean(times)))
```

D Siamese-FC code

D.1 Testing

```
import sys
import os
import glob
import numpy as np
from PIL import Image
sys.path.append('siamfc')
import src.siamese as siam
from src.tracker import tracker
from src.parse_arguments import parse_arguments
from src.region_to_bbox import region_to_bbox

def iou(pred_, gt_):
    pred = pred_
    gt = gt_
    pred[2] += pred[0]
    pred[3] += pred[1]
    gt[2] += gt[0]
    gt[3] += gt[1]

    x1s = np.fmax(pred[0], gt[0])
    x2s = np.fmin(pred[2], gt[2])
    y1s = np.fmax(pred[1], gt[1])
    y2s = np.fmin(pred[3], gt[3])
    ws = np.fmax(x2s - x1s, 0)
    hs = np.fmax(y2s - y1s, 0)
    intersection = ws * hs
    predArea = (pred[2] - pred[0]) * (pred[3] - pred[1])
    gtArea = (gt[2] - gt[0]) * (gt[3] - gt[1])
    union = np.fmax(predArea + gtArea - intersection, .00001)
    return intersection * 1.0 / union

if __name__ == '__main__':
    hp, evaluation, run, env, design = parse_arguments()
    final_score_sz = hp.response_up * (design.score_sz - 1) + 1
    filename, image, templates_z, scores = siam.build_tracking_graph(
                                            final_score_sz, design, env)

    frame_name_list = sorted(glob.glob('../data/*.jpg'))
    gt = open('../data/labels.txt').readlines()
    frame_name_list = frame_name_list[:20]
    gt = gt[:20]
    for i in range(len(gt)):
        gt[i] = np.array(gt[i].split(' ')[1:], float)
        gt[i][2] = gt[i][2]-gt[i][0]
        gt[i][3] = gt[i][3]-gt[i][1]
    n_frames = len(frame_name_list)

    with Image.open(frame_name_list[0]) as img:
        frame_sz = np.asarray(img.size)
        frame_sz[1], frame_sz[0] = frame_sz[0], frame_sz[1]

    continuous = True if '--continuous' in sys.argv else False
    starts = range(n_frames) if continuous else [0]
    n_subseqs = len(starts)
    times = []
    preds = []
    ious = []
    last_bbox = gt[0]

    print(n_subseqs)
    for i in range(n_subseqs):
        start_frame = starts[i]
```

```
        end_frame = start_frame+1 if continuous else n_frames
        gti = gt[start_frame : end_frame]
        frame_name_list_i = frame_name_list[start_frame : end_frame]
        pos_x, pos_y, target_w, target_h = gti[0] if not continuous else
                                                last_bbox

        bboxes, time = tracker(hp, run, design, frame_name_list_i, pos_x,
                                                pos_y, target_w, target_h,
                                                final_score_sz, filename,
                                                image, templates_z, scores,
                                                start_frame)
        preds.append(bboxes)
        times.append(time)
        ious += [iou(out, label) for out, label in zip(bboxes, gti)]
        last_bbox = bboxes[0]

    print("Average IOU: {:.2f}".format(np.mean(ious)))
    print("Average time: {:.2f}".format(np.mean(times)))
    print("Average FPS: {:.2f}".format(1.0/np.mean(times)))
```

E iOS app

E.1 Tracker

```
import Foundation
import UIKit
import CoreML

let PADDING: Double = 0.5
let INPUT_SIZE: Int = 227

class Tracker {
    let model: goturn

    var first_frame: UIImage?
    var last_frame: UIImage?
    var first_target: CVImageBuffer?
    var last_bbox: [Int]?

    var image_height: Int = 0
    var image_width: Int = 0

    var count: Int = 0
    var tracking = false


    required init() {
        print("Loading model...")
        model = goturn()
    }

    func startTracking(first_frame frame: UIImage, starting_bbox bbox: [Int])
                                                    {
        last_frame = frame
        last_bbox = bbox

        count = 0

        image_height = Int(frame.size.height * frame.scale)
        image_width = Int(frame.size.width * frame.scale)
    }

    func getCrops(last_frame: UIImage,
                  curr_frame: UIImage,
                  last_bbox bbox: [Int]) ->
        ((CVPixelBuffer, CVPixelBuffer), (MLMultiArray) -> [Int]) {
        var target, image: CVPixelBuffer

        let bbox_w = Double(bbox[2] - bbox[0])
        let bbox_h = Double(bbox[3] - bbox[1])

        var region: [Int] = [
            bbox[0] - Int(PADDING * bbox_w),
            bbox[1] - Int(PADDING * bbox_h),
            bbox[2] + Int(PADDING * bbox_w),
            bbox[3] + Int(PADDING * bbox_h)
        ]

        let pad = CGFloat([-region[0],
                           -region[1],
                           region[2]-image_width,
                           region[3]-image_height, 0].max()!)
        if pad > 0 {
            target = last_frame.imageWithInsets(insetDimen: pad).pixelBuffer
                                                        ()!
```

```
            image = curr_frame.imageWithInsets(insetDimen: pad).pixelBuffer()
                                                                    !
            for i in 0..<4 {
                region[i] += Int(pad)
            }
        } else {
            target = last_frame.pixelBuffer()!
            image = curr_frame.pixelBuffer()!
        }

        target = resizePixelBuffer(target, cropX: region[0], cropY: region[1]
                                                          ,
                                   cropWidth: region[2]-region[0],
                                   cropHeight: region[3]-region[1],
                                   scaleWidth: INPUT_SIZE,
                                   scaleHeight: INPUT_SIZE)!
        image = resizePixelBuffer(image, cropX: region[0], cropY: region[1],
                                  cropWidth: region[2]-region[0],
                                  cropHeight: region[3]-region[1],
                                  scaleWidth: INPUT_SIZE,
                                  scaleHeight: INPUT_SIZE)!

        let conv = {(r_bbox: MLMultiArray) -> [Int] in
            let pad = Int(pad)
            let w: Double = Double(region[2] - region[0])
            let h: Double = Double(region[3] - region[1])
            var c_bbox: [Double] = [
                r_bbox[[NSNumber(value: 0)]].doubleValue,
                r_bbox[[NSNumber(value: 1)]].doubleValue,
                r_bbox[[NSNumber(value: 2)]].doubleValue,
                r_bbox[[NSNumber(value: 3)]].doubleValue
            ]

            c_bbox[0] = c_bbox[0] / 10.0 * w
            c_bbox[1] = c_bbox[1] / 10.0 * h
            c_bbox[2] = c_bbox[2] / 10.0 * w
            c_bbox[3] = c_bbox[3] / 10.0 * h

            let res: [Int] = [
                Int(c_bbox[0]) + region[0] - pad,
                Int(c_bbox[1]) + region[1] - pad,
                Int(c_bbox[2]) + region[0] - pad,
                Int(c_bbox[3]) + region[1] - pad
            ]

            return res
        }

        return ((target, image), conv)
    }

    func track(frame curr_frame: UIImage) throws -> [Int] {

        let last_frame = self.last_frame!
        let last_bbox = self.last_bbox!

        var target: CVPixelBuffer
        var image: CVPixelBuffer
        var conv: (MLMultiArray) -> [Int]


        ((target, image), conv) = getCrops(last_frame: last_frame,
                                           curr_frame: curr_frame,
                                           last_bbox: last_bbox)

        if count == 0 && first_target == nil {
            first_target = target
        }

        let modelInput = goturnInput(target__0: target, image__0: image)
```

```
        let out = try self.model.prediction(input: modelInput).fc4__0
        let new_bbox = conv(out)

        self.last_bbox = new_bbox
        self.last_frame = curr_frame

        return new_bbox
```

E.2 UAV control

```
class FPVViewController: UIViewController,  DJIVideoFeedListener,
                                        DJISDKManagerDelegate,
                                        DJIBaseProductDelegate {

    @IBOutlet var fpvView: UIImageView!
    @IBOutlet var label: UILabel!
    @IBOutlet var log: UILabel!

    var frame: UIImage?
    var lastFrame: UIImage?

    let tapRec = UITapGestureRecognizer()

    var flightController: DJIFlightController?

    var videoPreviewer: VideoPreviewer?
    var videoExtractor: VideoFrameExtractor?

    var buffer: CVPixelBuffer?

    var curBbox: [Int]?

    var panGesture  = UIPanGestureRecognizer()

    var point: CGPoint?

    var last: DispatchTime?

    let maxVel: Double = 2.0

    let minSize: Float = 0.02

    let tracker = Tracker()

    override func viewDidAppear(_ animated: Bool) {
        super.viewDidAppear(animated)

        //VideoPreviewer.instance().setView(self.auxView)
        DJISDKManager.registerApp(with: self)
    }

    override func viewWillDisappear(_ animated: Bool) {
        super.viewWillDisappear(animated)

        self.videoExtractor = VideoPreviewer.instance().videoExtractor
        videoPreviewer?.setView(nil)
        DJISDKManager.videoFeeder()?.primaryVideoFeed.remove(self)
    }


    override func viewDidLoad() {
        super.viewDidLoad()
        tapRec.addTarget(self, action: #selector(self.tappedView))
        fpvView.addGestureRecognizer(tapRec)
```

```
        Timer.scheduledTimer(withTimeInterval: 0.01, repeats: true) { (timer)
                                                    in
            self.update()
        }
    }

    func tappedView(touch: UITapGestureRecognizer){
        //let point = touch.location(in: self.fpvView)

        tracker.tracking = false
        // (point - fpvView.origin) / fpvView.size
        let screenSize = UIScreen.main.bounds
        let screenWidth = screenSize.width
        let screenHeight = screenSize.height
        let fpvFrame: CGRect = self.fpvView!.frame
        let fpvX = fpvFrame.origin.x
        let fpvY = fpvFrame.origin.y
        let fpvWidth = fpvFrame.width
        let fpvHeight = fpvFrame.height

        curBbox = nil
        if point == nil {
            point = touch.location(in: nil)
        } else {
            var x1 = (point?.y)!
            var y1 = (point?.x)!
            point = touch.location(in: nil)
            var x2 = (point?.y)!
            var y2 = (point?.x)!

            x1 = ((x1 - fpvX) / fpvWidth) * frame!.size.width
            y1 = (1-((y1 - fpvY) / fpvHeight)) * frame!.size.height
            x2 = ((x2 - fpvX) / fpvWidth) * frame!.size.width
            y2 = (1-((y2 - fpvY) / fpvHeight)) * frame!.size.height


            self.curBbox = [Int(min(x1, x2)), Int(min(y1, y2)), Int(max(x1,
                                                    x2)), Int(max(y1, y2))]
            point = nil
        }
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
    }


    //
    //  DJIBaseProductDelegate
    //

    func productConnected(_ product: DJIBaseProduct?) {

        NSLog("Product Connected")

        if (product != nil) {
            product!.delegate = self

            VideoPreviewer.instance().start()
            self.videoPreviewer = VideoPreviewer.instance()


            let alert = UIAlertController(
                title: "Product Connected",
                message: "Product connected successfully",
                preferredStyle: UIAlertControllerStyle.alert)
            alert.addAction(UIAlertAction(
                title: "Ok", style:
                UIAlertActionStyle.default,
                handler: nil))
```

```
            self.present(alert, animated: true, completion: nil)

            flightController = (product as! DJIAircraft).flightController!
            flightController!.rollPitchControlMode =
                DJIVirtualStickRollPitchControlMode.velocity
            flightController!.yawControlMode =
                DJIVirtualStickYawControlMode.angularVelocity
        }
    }

    func productDisconnected() {
        NSLog("Product Disconnected")
        VideoPreviewer.instance().clearVideoData()
        VideoPreviewer.instance().close()
    }

    //
    //  DJISDKManagerDelegate
    //
    func appRegisteredWithError(_ error: Error?) {

        if (error != nil) {
            let alert = UIAlertController(title: "Register app failed",
                message: "Register app failed! Please enter "
                + "your app key and check the network.",
                preferredStyle: UIAlertControllerStyle.alert)
            self.present(alert, animated: true, completion: nil)
            alert.addAction(UIAlertAction(title: "Ok",
                                          style: UIAlertActionStyle.default,
                                          handler: nil))
        } else {
            let alert = UIAlertController(title: "Register app succeeded",
                message: "Register app succeeded.",
                preferredStyle: UIAlertControllerStyle.alert)
            alert.addAction(UIAlertAction(title: "Ok"
                style: UIAlertActionStyle.default,handler: nil))
            self.present(alert, animated: true, completion: nil)
        }
        DJISDKManager.startConnectionToProduct()
        DJISDKManager.videoFeeder()?.primaryVideoFeed.add(self, with: nil)
    }

    //
    //  DJIVideoFeedListener
    //
    func videoFeed(_ videoFeed: DJIVideoFeed,
            didUpdateVideoData rawData: Data) {
        let videoData = rawData as NSData
        let videoBuffer =
            UnsafeMutablePointer<UInt8>.allocate(capacity: videoData.length)

        videoData.getBytes(videoBuffer, length: videoData.length)
        VideoPreviewer.instance()
            .push(videoBuffer, length: Int32(videoData.length))
    }

    func update() {
        if last == nil {
            last = DispatchTime.now()
        }
        do {
            if let extractor = self.videoPreviewer?.videoExtractor {
                if let buffer = extractor.getCVImage() {
                    if var image = UIImage(
                        pixelBuffer: (buffer.takeRetainedValue())) {

                        self.frame = image
                        if let bbox = self.curBbox {
                            image = drawRectangleOnImage(image: image,
```

```
                                                    x: bbox[0],
                                                    y: bbox[1],
                                                    w: bbox[2]-bbox[0],
                                                    h: bbox[3]-bbox[1])
                        }
                        DispatchQueue.main.async {
                            self.fpvView.image = image
                        }

                        if let bbox = self.curBbox {
                            if lastFrame == nil {
                                lastFrame = frame
                            }
                            // track
                            if self.tracker.tracking {
                                self.curBbox =
                                    try tracker.track(frame: self.frame!)
                            } else {
                                self.tracker.startTracking(
                                    first_frame: frame!,
                                    starting_bbox: curBbox!)
                            }

                            lastFrame = frame

                            let now = DispatchTime.now()
                            let nanoTime = now.uptimeNanoseconds -
                                last!.uptimeNanoseconds
                            let fps = 1.0 / (Double(nanoTime) / 1e9)
                            self.label.text = String(fps)
                            last = now
                        }
                    }
                }
            }
            updateVel()
        } catch {
            print("ERROR")
        }
    }

    func setVel(pitch p: Float, roll r: Float) {
        if (self.flightController!.isVirtualStickControlModeAvailable()) {
            self.log.text = String(format: "pitch: %3.2f, roll: %3.2f", p, r)
        } else {
            flightController!.setVirtualStickModeEnabled(true)
            flightController!.setFlightOrientationMode(
                DJIFlightOrientationMode.aircraftHeading
            )

            self.log.text = "Error, control mode unavailable."
        }
        let controlData = DJIVirtualStickFlightControlData(pitch: p,
                                                           roll: r,
                                                           yaw: 0,
                                                           verticalThrottle:



        flightController!.send(controlData)
    }

    func updateVel() {
        if (self.flightController != nil) {
            if curBbox == nil {
                setVel(pitch: 0, roll: 0)
            } else {
                if let frame = self.frame {
                    let heightInPoints = frame.size.height
                    let height = heightInPoints * frame.scale
```

```
                    let widthInPoints = frame.size.width
                    let width = widthInPoints * frame.scale

                    if Float(abs(curBbox![2]-curBbox![0]))
                            / Float(width) < minSize
                            || Float(abs(curBbox![3]-curBbox![1]))
                            / Float(height) < minSize {
                        curBbox = nil
                        return
                    }

                    let centerX = Double(curBbox![0])
                        + Double(curBbox![2]-curBbox![0]) / 2.0
                    let centerY = Double(curBbox![1])
                        + Double(curBbox![3]-curBbox![1]) / 2.0

                    let relCenterX = centerX / Double(width)
                    let relCenterY = centerY / Double(height)

                    let velX = -(relCenterX - 0.5) * maxVel
                    let velY = -(relCenterY - 0.5) * maxVel

                    setVel(pitch: Float(velY), roll: Float(velX))
                }
            }
        }
    }
}
```

References

[1] H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. In A. Leonardis, H. Bischof, and A. Pinz, editors, *Computer Vision – ECCV 2006*, pages 404–417, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[2] L. Bertinetto, J. Valmadre, J. F. Henriques, A. Vedaldi, and P. H. Torr. Fully-convolutional siamese networks for object tracking. *arXiv preprint arXiv:1606.09549*, 2016.

[3] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[4] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111(1):98–136, jan 2015. DOI: 10.1007/s11263-014-0733-5.

[5] D. Gordon, A. Farhadi, and D. Fox. Re3 : Real-time recurrent regression networks for object tracking. *CoRR*, abs/1705.06368, 2017.

[6] C. Harris and M. Stephens. A combined corner and edge detector. In *In Proc. of Fourth Alvey Vision Conference*, pages 147–151, 1988.

[7] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[8] D. Held, S. Thrun, and S. Savarese. Learning to track at 100 FPS with deep regression networks. *CoRR*, abs/1604.01802, 2016.

[9] R. E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[10] M. Kristan, A. Leonardis, J. Matas, M. Felsberg, R. Pflugfelder, L. ehovin, T. Vojír, G. Häger, A. Lukei, G. Fernandez Dominguez, A. Gupta, A. Petrosino, A. Memarmoghadam, A. Garcia-Martin, A. Solís Montero, A. Vedaldi, A. Robinson, A. Ma, A. Varfolomieiev, and Z. Chi. The visual object tracking vot2016 challenge results. 9914:777–823, 10 2016.

[11] M. Kristan, J. Matas, A. Leonardis, M. Felsberg, L. Cehovin, G. Fernandez, T. Vojir, G. Hager, G. Nebehay, and R. Pflugfelder. The visual object tracking vot2015 challenge results. In *The IEEE International Conference on Computer Vision (ICCV) Workshops*, December 2015.

[12] M. Kristan, R. Pflugfelder, A. Leonardis, J. Matas, L. ehovin, G. Nebehay, T. Vojí, G. Fernandez Dominguez, A. Lukei, A. Dimitriev, A. Petrosino, A. Saffari, B. Li, B. Han, H. Cherkeng, C. Garcia, D. Pangeri, G. Häger, F. Shahbaz, and Z. Niu. The visual object tracking vot2014 challenge results. 09 2014.

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[14] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.

[15] D. G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the International Conference on Computer Vision-Volume 2 - Volume 2*, ICCV '99, pages 1150–, Washington, DC, USA, 1999. IEEE Computer Society.

[16] G. Ning, Z. Zhang, C. Huang, Z. He, X. Ren, and H. Wang. Spatially supervised recurrent convolutional neural networks for visual object tracking. *CoRR*, abs/1607.05781, 2016.

[17] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.

[18] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein, A. C. Berg, and F. Li. Imagenet large scale visual recognition challenge. *CoRR*, abs/1409.0575, 2014.

[19] M. J. Shafiee, B. Chywl, F. Li, and A. Wong. Fast YOLO: A fast you only look once system for real-time embedded object detection in video. *CoRR*, abs/1709.05943, 2017.

[20] A. W. M. Smeulders, D. M. Chu, R. Cucchiara, S. Calderara, A. Dehghan, and M. Shah. Visual tracking: An experimental survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(7):1442–1468, July 2014.

[21] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[22] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015.

[23] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013.